УДК 519.6: 004.942

С.В. ЧОПОРОВ
Запорожский национальный университет

ПРЕДСТАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СЕТОК В СИСТЕМАХ ИНЖЕНЕРНОГО АНАЛИЗА

Статья посвящена разработке формального представления дискретных моделей геометрических объектов для использования в системах инженерного анализа. Дискретные модели представлены множествами узлов и элементов, а также списками инцидентности. Разработанные представления не зависят от формы элементов и размерности моделей. Предложено две возможные реализации на основе векторных и списковых структур данных, для которых выполнены вычислительное исследование скорости выполнения операций добавления и удаления узлов или элементов.

Ключевые слова: дискретная модель, сетка, структура данных, формальное представление, быстродействие.

С.В. ЧОПОРОВ
Запорізький національний університет

ПОДАННЯ ОБЧИСЛЮВАЛЬНИХ СІТОК У СИСТЕМАХ ІНЖЕНЕРНОГО АНАЛІЗУ

Дискретні моделі геометричних об'єктів (або сітки) є важливими компонентами систем інженерного аналізу, заснованих на розв'язанні систем рівнянь у частинних похідних. Інтуїтивно, дискретна модель геометричного об'єкта – це колекція вузлів і елементів. Координати вузлів визначають геометрію об'єкта. Елементи визначають зв'язки між вузлами, тобто топологію геометричного об'єкта. У загальному випадку, елементи можуть мати тільки спільні вершини, ребра та грані. Для забезпечення чисельної збіжності для кожного внутрішнього ребра або грані одного елемента повинні існувати відповідне ребро або грань іншого. Отже, у статті розглянута проблема подання дискретних моделей геометричних об'єктів у системах інженерного аналізу.

У статті запропоновано підхід до подання дискретних моделей геометричних об'єктів множинами вузлів і елементів, а також списками інцидентності. Розроблений підхід до подання сіток не залежить від форми лементів і розмірності моделі. Для запропонованого підходу представлено алгоритми додавання та видалення вузлів або елементів.

Розглянуто дві можливі реалізації подання дискретних моделей геометричних об'єктів: на основі векторних і списковых структур даних. Векторні структури базуються на розміщенні даних у послідовних адресах пам'яті, що дозволяє ефективно їх обробляти у циклах. Натомість, спискові структури дозволяють ефективно видаляти дані. У статті наведено результати обчислювальних експериментів для дослідження швидкодії операцій ставки та видалення вузлів та елементів.

Розроблені класи для програмної реалізації подання дискретних моделей. Ці класи дозволяють використовувати простий і дружній до користувача інтерфейс для автоматизації алгоритмів генерації дискретних моделей.

Перспективи подальших досліджень можуть полягати у розробці подання дискретних моделей геометричних об'єктів у паралельних і розподілених комп'ютерних системах.

Ключові слова: дискретна модель, сітка, структура даних, формальне подання, швидкодія.

S.V. CHOPOROV
Zaporizhzhya National University

REPRESENTATION OF COMPUTATIONAL MESHES IN COMPUTER-AIDED ENGINEERING

A discrete model of a geometric object (aka grid, mesh) is an essential part of computer-aided engineering based on the solution of partial differential equations. Intuitively, a discrete model of geometric object is a collection of nodes and elements. Coordinates of nodes define the geometry of the object. Elements define the connectivity between nodes and the topology of the object. Generally, elements have only common vertices, edges and faces. To support computational convergence, for each internal edge or face of some element must exist the corresponding edge or face of the another element. In this paper, we present the problem of representation of discrete models of geometric models in computer-aided engineering.

In the paper, the approach for the discrete model representation based on sets of nodes, elements and lists of incidence has been developed. This approach does not depend on the element shape or the model dimension. For this representation, algorithms of node and element inserting and deleting have been described.

Two possible implementations of the developed representation have been compared. The first implementation uses list data structures, the second uses dynamic arrays. Lists allow effective deleting of nodes and elements by the modification of references only. Dynamic arrays store all data in contiguous block of memory and they are effective in searching and iterating procedures. Results of benchmarks for both implementations have been described in the paper.

Classes for both implementations have been developed. These classes allow using a simple user-friendly interface for meshing algorithms programming.

The further prospects of the research include the development of the representation adapted for parallel and distributed computer systems.

Keywords: discrete model, mesh, data structure, representation, benchmark.

Постановка проблемы

Модели, основанные на численном решении уравнений в частных производных, часто используют в задачах проектирования и исследования технических объектов. Один из наиболее распространенных на практике численных методов – метод конечных элементов – (как и некоторые другие) заменяет исходную непрерывную модель дискретной, которая может быть обработана компьютером. В результате на первом шаге моделирования для некоторой непрерывной области необходимо построить конечное множество простых фигур (элементов), например, треугольников, четырехугольников, тетраэдров, шестигранников и т. д. Взаимное расположение вершин элементов существенно влияет на точность численного анализа. Следовательно, автоматизация построения дискретных моделей геометрических объектов является одним из ключевых этапов реализации систем инженерного анализа.

Интуитивно, дискретная модель геометрического объекта – это коллекция узлов и элементов, которые определяет форму этого объекта. Координаты узлов определяют геометрию объекта, элементы – его топологию. В общем случае, элементы могут иметь только общие вершины, ребра и грани. При этом для каждой пары узлов, которая является ребром в одном элементе, должно существовать ребро в другом элементе, если оба элемента инцидентные в этих узлах. Аналогичное требование можно сформулировать относительно граней. Эти два требования необходимы для обеспечения непрерывности численного решения, соответственно, вдоль ребер и граней.

Формальное определение дискретной модели геометрического объекта – это описание соответствующей структуры данных. Оно необходимо с точки зрения, что пара алгоритм и структура данных является основной для создания программы. В то же время, разработка общей структуры данных для произвольной дискретной модели является весьма сложной задачей.

Анализ последних достижений и публикаций

Изначально наибольшее распространение получили полигональные дискретные модели поверхностей геометрических объектов. Для них разработаны весьма эффективные программные и аппаратные методы визуализации (например, при помощи обратной трассировки лучей). Многоугольник в порядке обхода однозначно определяется координатами своих вершин. Соответственно, в задачах визуализации для каждого полигона достаточно хранить только координаты его вершин. В результате на каждый полигон будет выделяться $3 \cdot k \cdot f$ байт памяти (k – число вершин, f – число байт, используемое для хранения одного числа с плавающей точкой).

В задачах генерации и обработки дискретных моделей возникает необходимость вставки или удаления узлов. В случае хранения элементов в виде массивов координат необходим поиск элементов с общими узлами. В результате в системах инженерного анализа распространение получили структуры данных, основанные на представлении вершин элементов указателями на позиции в общем массиве или списке координат вершин. При этом, как правило, элементы обладают дополнительной топологической информацией, упрощающей нахождение соседей. Пионером в этом направлении можно считать схему "крылатое ребро" (winged edge) [1]. В нем каждый узел хранит информацию о своих координатах и ссылку на одно из своих ребер, грань хранит ссылки на ребра в порядке их обхода. Ребро хранит ссылки на два узла и две грани (условно левую и правую), а также четыре "крыла": ссылки на предыдущие и следующие ребра в порядке обхода обеих граней. Схожие идеи заложены в схемы "радиальное ребро" (radial-edge) [2], "ориентированное ребро" (directed edge) [3], "полуребро" (half-edge) [4], а также обобщениях последнего [5, 6, 7]. Подобные схемы весьма эффективны при хранении и обработке полигональных моделей, но их весьма проблематично применять для объемных элементов. Компактные схемы хранения объемных элементов разработаны для тетраэдров в виде обобщения схем для треугольников [8, 9]. В то же время, в работе [10] предложена схема сжатого хранения дискретных моделей на основе шестигранников.

Цель исследования

Разработка систем инженерного анализа требует единой абстрактной структуры данных, независящей от формы элемента. Ее введение позволит использовать единую формальную нотацию при изложении алгоритмов. Таким образом, целью исследования является разработка схемы представления дискретных моделей геометрических объектов, а также соответствующих структур данных.

Изложение основного материала исследования

Формально, дискретная модель геометрического объекта – пара множеств вида

$$\mathbf{M} = (\mathbf{V}, \mathbf{E}), \quad (1)$$

где $\mathbf{V} = (v_1, v_2, \ldots v_m)$ – множество координат узлов; $\mathbf{E} = \left(e_1 = \left(k_{1,1}, k_{1,2}, \ldots, k_{1,n_1}\right), e_2 = \left(k_{2,1}, k_{2,2}, \ldots, k_{2,n_2}\right), \ldots, e_q = \left(k_{q,1}, k_{q,2}, \ldots, k_{q,n_q}\right)\right)$ – множество n_i-вершинных элементов ($n_i = |e_i|$) – кортежей номеров узлов из множества $\mathbf{V}$, определяющих положения вершин элемента; m – число узлов; q – число элементов.

Координаты j-го элемента с n вершинами определяются подстановкой $\mathbf{V}(e_j) = \left(v_{k_{j,1}}, v_{k_{j,2}}, \ldots, v_{k_{j,n}}\right)$.

Множество $\mathbf{V}$, в общем случае, можно считать ассоциативным массивом, в котором хранятся пары ключ-значение. Ключом выступает номер, код или адрес в памяти узла, значением – его координаты. Соответственно, запись $\mathbf{V}[k]$ обозначает координаты (радиус-вектор) узла $v_k \in \mathbf{V}$.

Аналогичное утверждение справедливо и для множества $\mathbf{E}$, в котором ключами могут выступать номера, коды или адреса элементов, а значениями соответствующие списки или массивы, хранящие кортежи. В таком случае запись $\mathbf{E}[i]$ обозначает элемент $e_i \in \mathbf{E}$. Учитывая, что e_i является кортежем, то $e_i[j] = \mathbf{E}[i][j]$ обозначает номер узла $k_{i,j}$, определяющего положение j-й вершины.

Алгоритмы вставки и удаления узлов и элементов для дискретной модели вида (1) являются элементарными. Добавление нового элемента требует поочередного добавления всех его вершин. В свою очередь, вставка нового узла требует проверки наличия узла с такими же координатами. Результатом операции вставки будет код (номер) узла в множестве $\mathbf{V}$ (новый или существующий, если такой же узел был найден). При удалении узла также необходимо будет удалить инцидентные элементы, а при удалении элемента необходимо удалить только узлы, на которые больше не участвуют в связях.

Для дискретной модели, состоящей из m узлов и q n-вершинных элементов, операция вставки узла может быть реализована при помощи линейного алгоритма с вычислительной сложностью $O(m)$. Добавление элемента, заданного координатами своих вершин, n раз потребует выполнения вставки узла и, соответственно, будет обладать вычислительной сложностью $O(nm)$. Для формирования множества инцидентных некоторому узлу элементов достаточно будет одного прохода по элементам множества $\mathbf{E}$, следовательно, вычислительная сложность этой процедуры будет равна q операций. В результате при удалении одного элемента потребуется nq операций, а при удалении узла $(kn+1)q$ операций (k – число инцидентных в узле элементов).

Очевидным недостатком дискретной модели геометрического объекта вида (1) является необходимость использования линейного поиска для формирования списков инцидентных элементов. Эти операции возникают при удалении узлов и элементов, а также распространены при обработке дискретных моделей (например, локальном сглаживании). Для избавления от этого недостатка можно предположить, что каждый узел кроме своих координат хранит еще информацию о множестве смежных элементов. В результате дискретную модель геометрического объекта можно представить в следующем виде.

$$\mathbf{M} = (\mathbf{N}, \mathbf{E}), \quad (2)$$

где $\mathbf{N} = \left((v_1, \mathbf{I}_1), (v_2, \mathbf{I}_2), \ldots (v_m, \mathbf{I}_m)\right)$ – множество пар координат i-го узла v_i и множеств инцидентных ему элементов $\mathbf{I}_i = \{e \in \mathbf{E}: \quad i \in E\}$; $\mathbf{E}$ – такое же, как в (1).

В квадратных скобках можно обозначить операции доступа к элементам множества $\mathbf{V}$, а в круглых скобках – к элементам множества $\mathbf{E}$. То есть, $\mathbf{M}[k] = \mathbf{N}[k] = v_k$, $\mathbf{M}(i) = \mathbf{E}[i]$, $\mathbf{M}(i,j) = \mathbf{E}[i][j]$. Для дискретной модели вида (2) к ним добавится операция получения множества инцидентных элементов: $\mathbf{M}.\text{incident}(k) = \mathbf{I}_k$. Также положим, что запись $\mathbf{M}.\text{code}(n)$ соответствует коду n в $\mathbf{M}$.

Вставка узла в дискретную модель вида (2) аналогична описанной выше. В результате вставка узла может быть формализована следующим алгоритмом.

algorithm insert_node

input:

$\mathbf{M}$ – дискретная модель геометрического объекта

p – координаты узла

output:

k – код узла

begin

for each $n \in \mathbf{M}.\mathbf{N}$ **do**

if $\|p - n.v\| < \varepsilon$ **then**

return $\mathbf{M}.\text{code}(n)$

end if

end for

$\mathbf{M}.\mathbf{N} \leftarrow \mathbf{M}.\mathbf{N} \cup \{(p, \varnothing)\}$

return $\mathbf{M}.\text{code}((p, \varnothing))$

Алгоритм вставки нового элемента, заданного списком координат своих вершин, в дискретную модель может быть формализован в виде

algorithm insert_element

input:

$\mathbf{M}$ – дискретная модель геометрического объекта

$\mathbf{V}$ – список координат вершин

begin

$(e, k_e) \leftarrow$ Новый элемент с $|\mathbf{V}|$ вершинами и его код в $\mathbf{M}.\mathbf{E}$

for each $v \in \mathbf{V}$ **do**

$e[i] \leftarrow \text{insert\_node}(\mathbf{M}, v)$

$\mathbf{M}.\text{incident}(e[i]) \leftarrow \mathbf{M}.\text{incident}(e[i]) \cup \{k_e\}$

end for

$\mathbf{M}.\mathbf{E} \leftarrow \mathbf{M}.\mathbf{E} \cup \{e\}$

end

Операция удаления элемента для дискретной модели геометрического объекта вида (2) требует обновления множеств инцидентных элементов в узлах, соответствующих вершинам удаляемого элемента. В результате эта операция может быть записана в виде следующего алгоритма.

algorithm remove_element

input:

$\mathbf{M}$ – дискретная модель геометрического объекта

k_e – код элемента

begin

$e \leftarrow \mathbf{M}(k_e)$

for $i = 1$ **to** $|e|$ **do**

$\mathbf{M}.\text{incident}(e[i]) \leftarrow \mathbf{M}.\text{incident}(e[i]) \setminus \{k_e\}$

if $\mathbf{M}.\text{incident}(e[i]) = \varnothing$ **then**

$\text{remove\_node}(\mathbf{M}, e[i])$

end if

end for

$\mathbf{M}.\mathbf{E} \leftarrow \mathbf{M}.\mathbf{E} \setminus \{e\}$

end

Операция удаления узла, используемая выше, может быть реализована в виде следующего алгоритма.

algorithm remove_node

input:

$\mathbf{M}$ – дискретная модель геометрического объекта

k_n – номер (код) узла

begin

$\mathbf{I} \leftarrow \mathbf{M}.\text{incident}(k_n)$

for each $k_e \in \mathbf{I}$ **do**

remove_element$(\mathbf{M}, k_e)$

end for

$\mathbf{M}.\mathbf{N} \leftarrow \mathbf{M}.\mathbf{N} \setminus \{\mathbf{M}.\mathbf{N}[k_n]\}$

end

Операции удаления узлов и элементов можно считать тесно связными друг с другом, т. к., вызов реализация одной из них зависит от реализации другой. Наличие взаимных вызовов в этих алгоритмах подчеркивает важность аккуратности их программной реализации.

Хранение дискретной модели геометрического объекта вида (2) возможно несколькими способами. Ключевыми отличиями между ними находятся в типах данных, используемых для хранения вершин и элементов. Традиционно, когда речь и идет о множествах однотипных экземпляров, противопоставляются две структуры данных: векторная и списковая.

Векторная структура данных базируется на размещении экземпляров в последовательных адресах памяти компьютера, что позволяет осуществлять доступ к элементу по его индексу за $O(1)$ времени. При этом удаление и вставка в середину новых экземпляров в такую структуру данных связано с линейными затратами времени, вызванными сдвигами массивов данных.

Списковая структура данных основана на включении дополнительно к информационным полям адресных полей-указателей, которые служат для определения связей в списке. Они позволяют быстро удалять и вставлять экземпляры в список, но приводят к уличению расхода памяти компьютера. Также их итерирование, как правило, является более медленным за счет большей фрагментации оперативной памяти.

Реализовать обе структуры можно в виде композиции трех классов: Mesh, Node и Element (рис. 1). Класс Mesh – абстракция сетки. Он является агрегацией экземпляров классов Node и Element и предоставляет интерфейс для манипулирования ими (добавления, удаления, создания итераторов). Класс Node является абстракцией узла и включает в себя (является композицией) координаты точки, тип узла (внутренний, граничный, характерный) и множество инцидентных элементов. Класс Element – абстракция элемента, которая хранит ссылки на узлы. При списковой реализации узлы и элементы хранятся в двусвязных списках, а в качестве ссылок (на узлы в элементах и на инцидентные элементы в узлах) используются соответствующие адреса памяти. При векторной – узлы и элементы хранятся в динамических массивах, а в качестве ссылок используются их индексы.

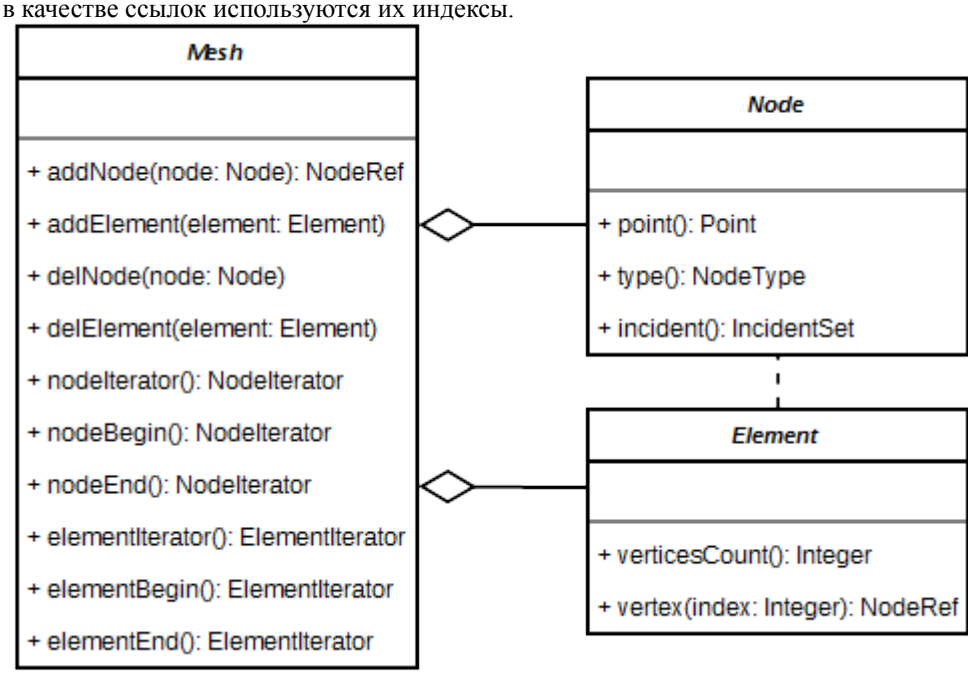

Рис. 1. Диаграмма классов

Рассмотрим две возможные реализации: 1 – основанную на использовании динамических массивов; 2 – основанную на использовании двусвязных списков для хранения узлов и элементов. Для исследования производительности воспользуемся модельными задачами: 1 – добавления произвольных набора восьми вершинных элементов, заданных координатами вершин; 2 – удаления произвольного набора узлов.

В качестве тестового используется компьютер с процессором Intel Core i5, 8 гигабайтами оперативной памяти, который находится под управлением операционной системы на базе Linux. Каждый тест запускается трижды, в результате фиксируется наименьшее время, потраченное на выполнение задачи.

На рис. 2 показана зависимость времени построения дискретной модели от числа элементов. Значения по оси абсцисс основное значение показывает число элементов в модели, а значения в скобках – узлов.

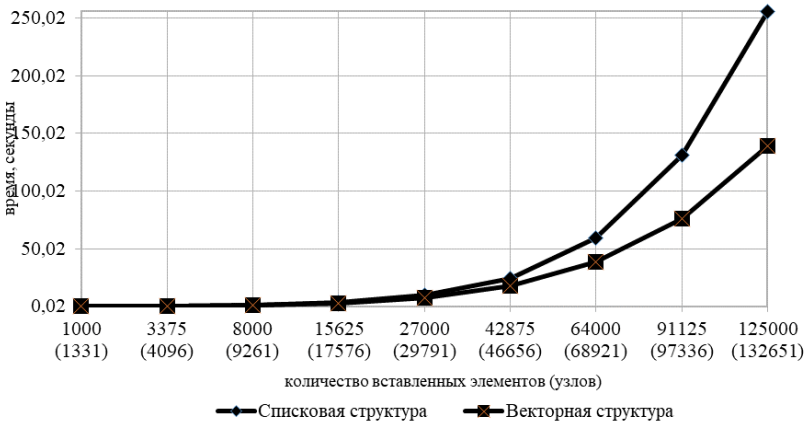

Рис. 2. Зависимость времени построения модели то числа элементов и узлов

Дискретная модель, основанная на динамических массивах (векторная структура), требует меньшего времени на формирование дискретной модели относительно аналогичной списковой. В обоих случаях графики имеют вид степенной функции. Вид графика обусловливается тем, что вычислительная сложность вставки n вершин с проверкой существования оценивается величиной $O(n^2)$.

На рис. 3 показана зависимость времени удаления тысячи узлов от общего их числа в дискретной модели геометрического объекта. Необходимость линейных сдвигов обусловливает существенно более медленное удаление (порядка 8 раз) при использовании векторных структур для хранения узлов и элементов в сравнении с списковыми. В обоих случаях зависимость времени от числа узлов в модели близка к линейной.

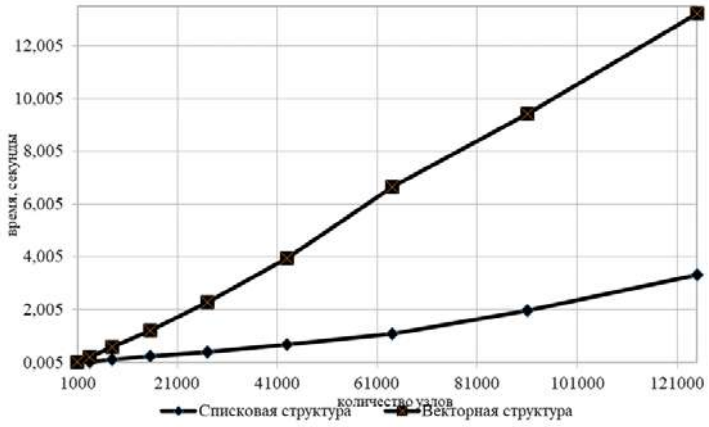

Рис. 3. Зависимость времени удаления 1000 узлов от числа узлов в модели

Для исследования влияния числа удаляемых узлов на время работы рассмотрим дискретную модель (рис. 4), состоящую из 27000 узлов и 24389 элементов. Как и в предыдущем случае, удаление узлов при использовании списковых структур для хранения узлов и элементов существенно быстрее векторных.

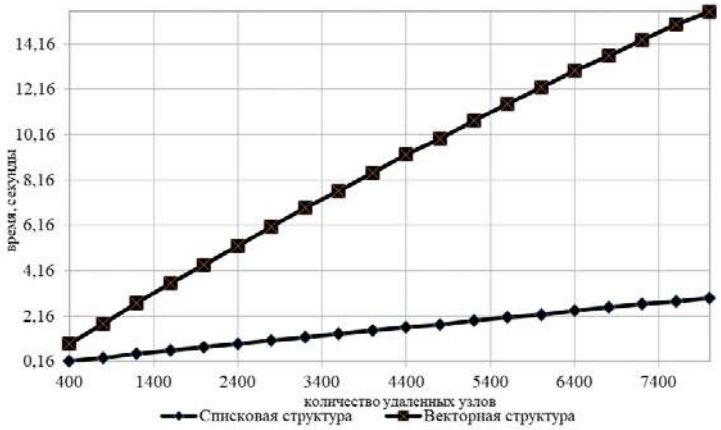

Рис. 4. Зависимость времени от числа удаленных узлов

В то же время, необходимо отметить, что операции удаления узлов и элементов при построении используются реже вставки и итерирования узлов, в которых более эффективными показали себя векторные структуры данных.

Выводы

Итак, основным научным результатом исследования является подход к формальному описанию дискретных моделей геометрических объектов, который позволяет использовать единую нотацию при изложении методов их автоматической генерации. Хранение вместе с координатами списка инцидентных элементов каждого узла позволяет понизить порядок сложности алгоритмов вставки и удаления узлов.

Серия вычислительных экспериментов для сравнения двух реализаций: на основе списковых и векторных структур данных показала, что первые более эффективны при удалении узлов, а вторые – при безопасном добавлении.

Более высокая эффективность списковых структур при удалении обосновывается тем, что удаление одного экземпляра со списка обладает вычислительной сложностью порядка $O(1)$. Недостатком таких структур является отсутствие возможности произвольного доступа (в общем случае доступ к экземпляру по индексу в списке требует итерирования списка).

Векторные структуры располагают экземпляры в последовательных адресах памяти компьютера и позволяют осуществлять произвольный доступ к экземпляру. В результате алгоритмы линейного поиска, используемые при вставке для поиска одинаковых вершин, работают быстрее (в том числе за счет более эффективного использования кэшей процессора). В результате операции вставки в 1,2–2 раза быстрее, чем для списковых. В то же время, векторные структуры практически на порядок медленнее списковых при удалении узлов и элементов модели, что объясняется необходимостью сдвигов массивов данных и обновления индексов. Однако векторные структуры позволяют эффективно распараллелить их итерирование, в том числе при обновлении индексов.

Список использованной литературы

1. Baumgart B. G. A polyhedron representation for computer vision / B. G. Baumgart // National Computer Conference and Exposition AFIPS’75 : New York, USA, 1975 : proceedings. – ACM, 1975. – P. 589–596.
2. Muuss M. J. Combinatorial solid geometry, boundary representations, and non-manifold geometry / M. J. Muuss, L. A. Butler // State of the Art in Computer Graphics: Visualization and Modeling. – 1991. – Volume 1. P. 185–223.
3. Campagna S. Directed Edges: A Scalable Representation for Triangle Meshes / S. Campagna, L. Kobbelt, H.-P. Seidel // Journal of Graphics Tools. – 1998. – Voluma 3, Issue 4. – P. 1–11.
4. Kettner L. Using generic programming for designing a data structure for polyhedral surfaces / L. Kettner // Computational Geometry: Theory and Applications. – 1999. – Volume 13, Issue 1. – P. 65–90.
5. Alumbaugh T. J. Compact array-based mesh data structure / T. J. Alumbaugh, X. Jiao // The 14th International Meshing Roundtable : International Conference, San Diego, California, USA, September 11–14, 2005 : proceedings. – Sandia National Laboratories: Springer-Verlag, 2005. – P. 585–504.
6. Levy B. Circular incident edge list: a data structure for rendering complex unstructured grids / B. Levy, G. Caumon, S. Conreaux, X. Cavin // Conference on Visualization : 2001, Washington, DC, USA :IEEE Computer Society, 2001. – P. 191–198.
7. Sieger D. Design, implementation, and evaluation of the surface_mesh data structure / D. Sieger, M. Botsch // The 20th International Meshing Roundtable : International Conference, 2011 : proceedings. – Sandia: Sandia National Laboratories, 2011. – P. 533–550.

8. Blandford D. K. Compact representation of simplicial meshes in two and three dimensions / D. K. Blandford, G. E. Blelloch, D. E. Cardoze, C. Kadow // International Journal of Computational Geometry and Applications. – 2005. – Volume 15, Issue 1. – P. 3–24.
9. Lage M. Chf: A scalable topological data structure for tetrahedral meshes / M. Lage, T. Lewiner, H. Lopes, L. Vehlo // The XIII Brazilian Symposium on Computer Graphics and Image Processing : Washington, DC, USA, 2005 : proceedings. – IEEE Computer Society, 2005. – P. 349–356.
10. Isenburg M. Compressing Hexahedral Volume Meshes / M. Isenburg, P. Alliez // Graphical Models. – 2003. – Volume 65, Issue 4. – P. 239–257.