Восстановление Информации о Смежности Вершин для Полигональных Сеток, Полученных с Помощью Метода Марширующих Кубов

Ростислав Хлебников
Institute for Computer Graphics and Vision
Graz University of Technology, Graz, Austria
khlebnikov@icg.tugraz.at

Аннотация

Topological information is crucial for many surface mesh processing algorithms. This information can be both desirable, e.g. for optimizing mesh rendering, and required, e.g. for surface curvature computation.

In this paper we propose a high-performance and robust algorithm for extracting mesh connectivity information for isosurfaces of 3D data that have been extracted using marching cubes algorithm.

Keywords: *Mesh topology, Marching Cubes, isosurface.*

1. ВВЕДЕНИЕ

Несмотря на существенный рост доступных вычислительных мощностей, извлечение изоповерхностей из пространственных данных по-прежнему требует значительных временных затрат. Кроме этого, увеличивается и объем данных, поступающих на вход алгоритмов. Так, современный компьютерный томографический сканер, способен создать объемное изображение размером 512x512x320 вокселей всего лишь за одно вращение [1]. Поэтому для извлечения изоповерхностей используются вычислительные возможности графических ускорителей, которые обеспечивают высокую степень параллелизации обработки данных. Так как метод марширующих кубов [2] является одним из наиболее распространенных методов построения изоповерхностей в силу его высокого быстродействия и простоты реализации, на данный момент уже описаны способы использования графических адаптеров для его ускорения [3].

Недостатком такого подхода является то, что полученная изоповерхность описывается лишь с помощью набора треугольников. Однако, для работы многих алгоритмов необходимо наличие топологической информации, как, например, для расчета кривизны [4], для сглаживания поверхности [5] или для автоматического расчета уровней детализации [6]. Таким образом, встает задача быстрого и надежного восстановления информации о топологии поверхностей, описанных с помощью набора треугольников.

Описанный в последующих секциях алгоритм использует небольшую модификацию алгоритма марширующих кубов для извлечения изоповерхности из пространственной регулярной решетки и сохраняет информацию о смежности вершин этой изоповерхности в компактной структуре данных. Каждая секция начинается с краткого обзора литературы в соответствующей области, за которым следует описание наших результатов.

2. СТРУКТУРА ДАННЫХ

На текущий момент предложено множество структур данных, позволяющих в компактном виде представить информацию о топологии полигональной сетки. Многие из них являются модификациями представления с помощью «крылатых» ребер (winged edge data structure, [7]). Так, например, представление с помощью радиальных ребер [8] пригодно также для описания поверхностей, не являющихся многообразиями. Кроме того весьма распространено описание полигональных сеток с помощью полу-ребер [9], которое реализовано в таких библиотеках, как CGAL [10] и OpenMesh [11].

Так как нам необходима информация лишь о смежности вершин, мы предлагаем структуру данных, основанную на представлении с помощью радиальных ребер, но с исключением излишней информации (см. Рис. 1).

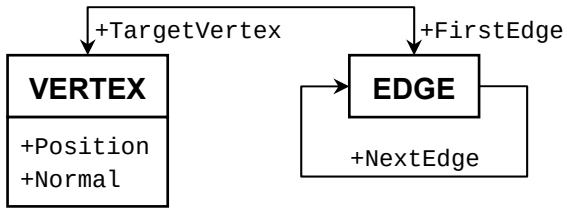

Рисунок 1: *Диаграмма классов структуры данных для определения смежных вершин.*

3. АЛГОРИТМ

При восстановлении топологической информации о полигональной сетке, описанной с помощью набора треугольников, одной из основных задач является определение того, какие из вершины, принадлежащие различным треугольникам, совпадают. Эта задача осложняется тем, что при вычислении координат вершины в алгоритме, создающем изначальное описание полигональной сетки, возможны погрешности вычислений, обусловленные использованием чисел с плавающей запятой. В работе [12], Рок и Вожный используют сбалансированное дерево двоичного поиска (а именно, АВЛ-дерево [13]) для нахождения и слияния пространственно близких вершин. В работе [14], МакМэйнс *и др.* используют хэш-таблицы для поиска одинаковых вершин. Это позволяет достигать более высокой производительности, однако, в силу особенностей хэш-функции, слияние вершин возможно только при одинаковом бинарном представлении координат этих вершин.

В качестве входных данных, оба вышеописанных метода используют файлы в формате STL. Эти файлы могут быть получены с помощью множества прикладных программ и поэтому, при создании алгоритмов восстановления топологической информации, невозможно полагаться на какие-либо дополнительные предположения о свойствах

входных данных. Однако, так как мы рассматриваем полигональные секти, созданные с помощью метода марширующих кубов, мы можем объединить положительные стороны алгоритмов, описанных выше.

Как и в [14], мы используем хэш таблицу для слияния вершин. Для того, чтобы избавиться от влияния численной погрешности расчетов при построении хэш-ключа, для каждой вершины, создаваемой при построении изоповерхности, мы записываем индекс ребра регулярной решетки, на котором создается эта вершина или индекс узла в случае если изоповерхность проходит непосредственно через узел решетки. Так как алгоритм марширующих кубов [2] создает не более одной вершины на каждом ребре решетки или проходит через ее узлы, этот индекс (а значит и значение хэш-функции) будет совпадать только у вершин, которые должны быть слиты. Точные координаты таких вершин, тем не менее, могут отличаться из-за погрешностей вычислений при обработке различных ячеек, имеющих общее ребро.

Для регулярных сеток размером до 1024^3, 32-битный индекс может быть построен, например, следующим образом:

$$Index = i + n_x \cdot \left(j + n_y \cdot (k + dirIndex \cdot n_z)\right),$$

где (i, j, k) – индекс узла, откуда исходит ребро, (n_x, n_y, n_z) – количество узлов регулярной сетки по каждой из координатных осей, а *dirIndex* равен 0, если изоповерхность проходит через этот узел, 1 для ребер, ориентированных по оси X, 2 – для оси Y и 3 – для оси Z.

В остальном, наш алгоритм следует схеме восстановления топологической информации в оперативной памяти, описанной в [14].

4. РЕЗУЛЬТАТЫ И ВЫВОДЫ

Мы реализовали предложенный алгоритм на языке C++. В качестве реализации алгоритма марширующих кубов мы использовали реализацию, распространяемую вместе с nVidia CUDA SDK [15]. В качестве хэш-функции мы использовали MurmurHash2A [16].

Мы сравнили результаты работы нашего алгоритма с алгоритмами, описанными в [12] и [14]. Результаты сравнения представлены в Таблице 1.

Количество вершин (тыс.)	**Предложенный алгоритм**	**Алгоритм** [14]	**Алгоритм** [12]
10	0.86	0.73	1.10
100	1.62	1.64	2.74
1000	4.05	4.02	6.52

Таблица 1: *Сравнение времени работы алгоритмов восстановления топологической информации (в секундах). Каждое время работы получено усреднением за 50 запусков. Алгоритм [14] оставляет неслитыми в среднем от 0.5% до 1.6% от общего числа вершин. Предложенный нами алгоритм и [12] создают абсолютно идентичные полигональные сетки.*

5. ЗАКЛЮЧЕНИЕ

В данной статье мы представили алгоритм, позволяющий надежное и быстрое восстановление топологической информации для полигональных сеток, полученных с помощью метода марширующих кубов, а также описали компактную структуру данных, позволяющую извлекать информацию о смежных вершинах в полигональной сетке.

Благдарности

Данная работа была спонсирована Европейским Союзом под инициативой FP7 VPH, контракт номер 223877.

6. СПИСОК ЛИТЕРАТУРЫ

[1] L.K. Anderson, “What’s Next for CT,” May. 2007.

[2] W.E. Lorensen and H.E. Cline, “Marching cubes: A high resolution 3D surface construction algorithm,” *Proceedings of the 14th annual conference on Computer graphics and interactive techniques - SIGGRAPH '87*, 1987, pp. 163-169.

[3] C. Dyken, G. Ziegler, C. Theobalt, and H. Seidel, “High-speed Marching Cubes using HistoPyramids,” *Computer Graphics Forum*, vol. 27, 2008, pp. 2028-2039.

[4] Zhong Li, B. Barsky, and Xiaogang Jin, “An effective third-order local fitting patch and its application,” *2009 IEEE International Conference on Shape Modeling and Applications*, Beijing, China: 2009, pp. 7-14.

[5] Z. Lv and X. Chen, “A Three Dimensional Mesh Improvement Algorithm Based on Curvature Flow,” *2009 Second International Symposium on Information Science and Engineering*, Shanghai, China: 2009, pp. 36-40.

[6] M. Hussain, Y. Okada, and K. Niijima, “LOD modelling of polygonal models based on multiple choice optimisation,” *10th International Multimedia Modelling Conference, 2004. Proceedings.*, Brisbane, Qld., Australia: , pp. 203-210.

[7] B.G. Baumgart, “A polyhedron representation for computer vision,” Anaheim, California: 1975, p. 589.

[8] K. Weiler, “The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Boundary Modeling,” *Geometric Modeling for CAD Applications*, Amsterdam: North-Holland, 1988, pp. 3-36.

[9] M. Mäntylä, *An introduction to solid modeling*, Rockville Md.: Computer Science Press, 1988.

[10] L. Kettner, “Halfedge Data Structures,” *CGAL User and Reference Manual*, CGAL Editorial Board, 2010.

[11] M. Botsch, S. Steinberg, S. Bischoff, and L. Kobbelt, “OpenMesh - a generic and efficient polygon mesh data structure,” 2002.

[12] S.J. Rock and M.J. Wozny, “Generating Topological Information from a “Bucket of Facets”,” 1992, pp. 251–259.

[13] Г.М. Адельсон-Вельский и Е.М. Ландис, “Один алгоритм организации информации,” *Доклады АН СССР*, Т. 146, №2, 1962, стр. 263–266.

[14] S. McMains, J.M. Hellerstein, and C.H. Séquin, “Out-of-core build of a topological data structure from polygon soup,” Ann Arbor, Michigan, United States: 2001, pp. 171-182.

[15] “NVIDIA CUDA SDK - Physically-Based Simulation,” `http://www.nvidia.com/content/cudazone/cuda_sdk/Physically-Based_Simulation.html`, 2008.

[16] A. Appleby, “Murmur hash,” `http://sites.google.com/site/murmurhash/`, 2009.