6. Scargle J., Jackson B., Norris J. Adaptive Piecewise-constant Modeling of Signals in Multidimensional Spaces // In proc. of the Statistical Problems in Particle Physics, Astrophysics and Cosmology, SLAC, Stanford, California, Sep. 8-11, 2003. P. 157–162.

7. Kawahara Y, Sugiyama M. Change-Point Detection in Time-Series Data by Direct Density-Ratio Estimation // In Proc. of the SIAM Intern. Conf. on Data Mining, SDM, April 30-May 2, 2009. Sparks.

8. Turner R., Saatei Y, Rasmussen C. E. Adaptive Sequential Bayesian Change Point Detection // Temporal Segmentation Workshop at NIPS. 2009. Dec. P. 1–4.

9. A Symbolic Representation of Time Series, with Implications for Streaming Algorithms / J. Lin, E. Keogh, S. Lonardi, B. Chiu // In proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, California. 13th June, 2003. P. 2-11.

10. Experiencing SAX: a Novel Symbolic Representation of Time Series / J. Lin, E. Keogh, S. Lonardi, B. Chiu // Data Mining and knowledge discovery. 2007. Vol. 15, № 2. P. 107–144.

11. Lin J., Keogh E., Fu. A.W. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence // In proc. of the 5th IEEE Intern. Conf. on Data Mining, Houston, TX, Nov. 27-30, 2005. P. 226–233.

12. Keogh E., Chakrabarti K., Pazzani M. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases // In proc. of ACM SIGMOD Conf. on Management of Data, Santa Barbara, May 21-24, 2001. P. 151–162.

13. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases / E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra // J. of Knowledge and Information Systems. 2001. Vol. 3. P. 263–286.

14. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. АН СССР. 1965. С. 845–848.

15. Гасфилд Д. Строки, деревья и последовательности в алгоритмах: Информатика и вычислительная биология / Пер. с англ. И. В. Романовского. СПб.: Невский Диалект; БХВ-Петербург, 2003.

16. Wagner R. A., Fischer M. J. The string-to-string correction problem // J. of the ACM. 1974. Vol. 21, № 1. P. 168–173.

N. A. Zhukova, I. S. Sokolov, A. V. Ekalo

THE METHOD OF FUZZY COMPARISON OF TELEMETRIC PARAMETERS BASED ON DATA MINING

This paper offers the method of fuzzy comparison of slowly changing telemetric parameters based on symbolical representation with use of the weighed edit distance. Application of edit distance allows to compare parameters displaced from each other by time and/or values, and to eliminate insignificant deviations in parameters behavior, caused by influence of external factors.

Data Mining, symbolical representation, telemetric parameters

004.42.032.24

М. А. Фирсов, С. А. Ивановский

Параллельная реализация алгоритма построения пересечения простых полигонов с использованием технологии CUDA

Описана параллельная реализация на CUDA модифицированного расширенного алгоритма Грейнера–Хормана построения пересечения простых плоских полигонов. Приведены результаты экспериментального сравнения времён выполнения последовательной версии и версии на CUDA.

Вычислительная геометрия, пересечение простых полигонов, технология CUDA

Операция построения оверлеев (объединения, пересечения или разности) плоских полигонов является одной из основных в системах автоматизированного проектирования (САПР), ГИС в других графических системах. На основе построения оверлеев решается целый ряд других смежных задач [1].

Разработано множество последовательных алгоритмов пересечения полигонов [2], но пока мало адаптаций этих алгоритмов для выполнения на графических ускорителях. В данной статье описывается параллельная версия для CUDA модифи-

цированного расширенного алгоритма Грейнера–Хорманна построения пересечения двух простых плоских полигонов [3]. Этот алгоритм был выбран по следующим причинам:

– в [4] показана возможность успешного распараллеливания алгоритма пересечения множества полигонов (получено ускорение в 20…40 раз над последовательной версией для CPU);

– относительная простота алгоритма;

– основная работа 1-й фазы алгоритма Грейнера–Хорманна – поиск всех пересечений ребер полигонов – используется во многих других алгоритмах пересечения полигонов.

При разработке параллельной реализации использован подход, аналогичный описанному в [4]. При этом 1-я фаза алгоритма распараллелена практически так же, как в [4]; 2-я фаза – по-другому; 3-я, как и в [4], оставлена для последовательного исполнения на CPU, так как она вычислительно простая и последовательна по природе.

Модифицированный расширенный алгоритм Грейнера–Хорманна. Расширенный алгоритм Грейнера–Хорманна подробно описан в [3] и состоит из трех фаз.

Фаза 1. Нахождение всех точек пересечения рёбер полигонов. Эти точки («вершины-пересечения») вставляются в соответствующие места списков вершин исходных полигонов и снабжаются флагом пересечения и указателем на вершину-двойника списка другого полигона.

Фаза 2. Производится обход полигона, и встреченные точки с флагом пересечения помечаются флагом входа (en), выхода (ex), входо-выхода (en-ex) либо выходо-входа (ex-en). Значение флага, сопоставляемое точке пересечения, определяется характеристиками предшествующего и следующего рёбер (точнее, рёберного фрагмента) для этой точки при обходе. Для ребра возможно 3 случая (3 типа принадлежности другому полигону):

– out – ребро лежит вне другого полигона;

– on – ребро совпадает с ребром другого полигона;

– in – ребро лежит внутри другого полигона.

В табл. 1 указано, какой флаг следует сопоставить точке пересечения в зависимости от характеристик предшествующего (prev) и последующего (next) рёбер.

Таблица 1

Ребро prev	Ребро next		
	out	on	in
out	enex	en	en
on	ex	none	en
in	ex	ex	exen

При определении принадлежности ребра тип on устанавливается просто (достаточно проверить, что его концы – вершины пересечения, а их двойники – соседи в списке вершин другого полигона). Если ребро не имеет тип on, то для определения принадлежности требуется проводить тест на попадание середины ребра внутрь другого полигона.

Фаза 3. Производятся обходы по спискам вершин исходных полигонов, дополненным вершинами-пересечениями, и получаются полигоны-пересечения.

Модифицированный расширенный алгоритм отличается от описанного в [3] некоторыми упрощениями, а также особенностями выполнения фазы 3.

Версия алгоритма для CUDA. Списки, использовавшиеся для представления полигонов в последовательном алгоритме, плохо подходят для обработки на GPU. С учётом особенностей доступа к глобальной памяти в CUDA для представления множества вершин используется набор одномерных массивов:

```
struct ArPolygon {
COORDTYPE* xx;//массив x-координат
COORDTYPE* yy;//массив y-координат
UINT* an;//массив номеров (номерных значений)
double* alfas;//массив значений alfa, характеризующих положение на ребре
char* secflgs;//массив флагов
UINT* dvoiniks;//массив индексов (указателей) вершин-двойников другого полигона
UINT N;//число вершин в полигоне }.
```

В последовательности вершин *i*-я точка представляется совокупностью *i*-х элементов нескольких массивов. Так как полигон – замкнутая линия, полезно иметь копию вершины 0 в конце массива.

Изначально в массиве номеров содержатся последовательные числа {0, 1, 2,…, $N-1$, 0}, которые можно считать номерами соответствующих исходных рёбер. Когда в полигон будут добавлены вершины-пересечения, то они получат номера

тех исходных рёбер, на которых располагаются (таким образом, в массиве номеров могут быть одинаковые числа). Место для вставки вершины-пересечения (рисунок) определяется по номерным значениям и значениям параметра alfa ∈ [0, 1).

Элемент массива флагов может принимать следующие значения:

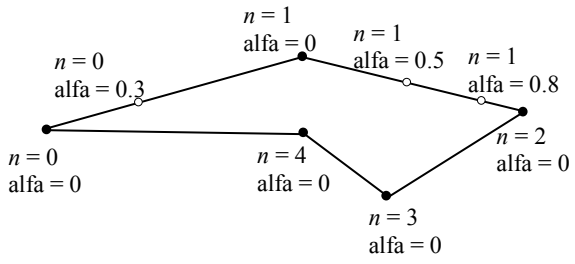
0 – вершина не является пересечением;

1 – вершина-пересечение с неизвестным флагом или с флагом none;

2, 3, 4, 5 – вершины-пересечения с флагами en, ex, enex и exen.

Совокупность массивов структуры ArPolygon назовем мультимассивом вершин.

Распараллеливание 1-й фазы алгоритма. Требуется найти все точки пересечения рёбер полигонов, установив у точек-двойников ссылки друг на друга. Точки пересечения разбивают исходные рёбра на рёберные фрагменты и становятся вершинами-пересечениями. Их следует добавить в полигоны, но добавлять вершины в массивы по отдельности крайне неэффективно. Поэтому в параллельной версии вершины-пересечения добавляются в полигоны не по одной, как в последовательной версии, а все вместе (после того, как все они будут найдены). При этом выполняется сортировка массива вершин, чтобы новые вершины заняли правильные места. Ключ сортировки – совокупность номерного значения и значения alfa (рисунок). Для сортировки использовалась самостоятельная реализация алгоритма быстрой сортировки на CUDA [5], при этом сначала упорядочивались ключи сортировки, а затем соответствующие перемещения выполнялись с самими вершинами.



Фаза 1 распараллелена в соответствии с идеями, изложенными в [4] (для алгоритма Грейнера–Хорманна). Пусть пересекаемые полигоны – А и Б. Тогда в фазе 1 выполняются следующие шаги:

1. Сделать копии исходных мультимассивов вершин. Эти копии понадобятся на следующей фазе 2.

2. Найти все пересечения рёбер и добавить их в мультимассивы полигонов А и Б.

3. Отсортировать мультимассив А.

4. Восстановить ссылки вершин мультимассива Б на вершины мультимассива А.

5. Отсортировать мультимассив Б.

6. Восстановить ссылки вершин мультимассива А на вершины мультимассива Б.

На шаге 2 используется метод двойного прохода – приём разработки параллельных алгоритмов, когда на первом проходе нити собирают информацию (каждая нить подсчитывает число найденных пересечений), а на втором – выполняют реальную работу: в мультимассивы добавляют пересечения (они находятся заново) в места, определённые с помощью первого прохода.

Распараллеливание 2-й *фазы алгоритма*. В [4] флаги вершин-пересечений предлагается устанавливать через выполнение префиксной суммы по вершинам-пересечениям, используя свойство чередования вершин-входов и вершин-выходов. В расширенной версии алгоритма это свойство может нарушаться, поэтому для определения флага вершины-пересечения требуется проверить принадлежность двух рёберных фрагментов к внутренней области или границе другого полигона (для этого лучше использовать его копию, сохранённую в начале фазы 1). Проверка принадлежности множества рёберных фрагментов – это работа, распараллеливаемая довольно очевидным образом, но всё же возникает немало вариантов. Далее приводятся особенности использовавшегося решения.

Каждой вершине-пересечению соответствует ровно один рёберный фрагмент, образованный этой вершиной-пересечением и её соседом справа (вершина является соседом справа, если расположена в массиве следующей). Определение принадлежности всех этих рёберных фрагментов требуется для расстановки флагов всем вершинам-пересечениям и составляет основную работу фазы 2. Флаги нужно проставить только вершинам-пересечениям, из остальных вершин будут использоваться лишь те, которые являются соседями справа для вершин-пересечений. Чтобы ненужные на фазе 2 вершины не мешали параллельной работе нитей, сначала составляется массив индексов (индексный массив) вершин-пересечений, во время прохода по которому и будет далее выполняться анализ рёбер, расстановка флагов.

Определение типов принадлежностей «OUT» и «IN» выполняется, как и в последовательной версии, с помощью алгоритма трассировки луча [6], применяемого к середине рёберного фрагмента и копии другого полигона. Каждая нить выполняет алгоритм трассировки луча для своего рёберного фрагмента, при этом нити, которым достались ON-рёбра, определяют это и алгоритм не выполняют, а лишь загружают данные для других нитей.

Пусть ставятся флаги пересечениям полигона А, при этом используется копия полигона Б; пусть в блоке имеются p нитей. Блок работает следующим образом:

1. Каждая нить проверяет, является ли соответствующее ей ребро ON-ребром.

2. Каждая нить загружает в разделяемую память координаты одной из вершин копии полигона Б.

3. Нити выполняют алгоритм трассировки луча, используя все загруженные вершины.

4. Если в копии полигона Б ещё остались вершины, не использовавшиеся в алгоритме трассировки луча, то переход на шаг 2 (повторяются шаги 2 и 3 для новой порции вершин копии полигона Б).

5. Нити обмениваются результатами для своих рёбер с соседними нитями, ставят флаг своей вершине-пересечению полигона А и, если возможно (см. далее), её двойнику.

6. Если в полигоне А ещё есть необработанные вершины пересечения, то переход на шаг 1 (повторяются шаги 1–6 для новой порции вершин-пересечений полигона А).

7. Установка флагов для крайних вершин-пересечений (подробности далее).

Для установки флагов крайних вершин-пересечений требуется информация (признак принадлежности левого инцидентного ребра) от соседнего блока. С помощью атомарных функций блоки синхронизируются со своими блоками-соседями, и те блоки, которые закончили работу раньше, оставляют информацию для тех блоков, которые закончат работу позже, чтобы последние поставили флаги для крайних вершин. Флаг крайней левой вершине устанавливает в итоге либо блок, которому она назначена, либо соседний слева блок. Для 0-го блока соседним слева является последний блок.

В описании шага 5 упоминается, что не всегда нить может поставить флаг двойнику обработанной вершины-пересечения. Если вершине-пересечению поставлен флаг ex или en, то двойник получает противоположный флаг, но если вершине-пересечению поставлен флаг exen или enex, то для определения флага двойника требуется дополнительная обработка. Для дополнительной обработки составляется индексный массив вершин полигона Б, которые всё ещё нуждаются в постановке флага, затем по индексному массиву выполняется их обработка. Она состоит в том, что нужно определить принадлежность (относительно копии полигона А) одного инцидентного ребра для каждой вершины, после чего возможно будет установить их флаги. Эта дополнительная обработка проще основной, так как на этот раз для установки флага не требуется информации от соседней нити или блока.

Результаты экспериментов.

Конфигурация тестовой системы:

Видеокарта: GeForce GTX 550 Ti 1024 MB.

Версия CUDA: 5.0.35.

Процессор: Intel Core i5-2400 3.1 GHz.

Компьютер: ОП – 3.49 Гб DDR3 1333 MHz (двухканальная), ОС – Windows XP SP3 32 bit.

В версии для CUDA время перемещения данных из ОП в глобальную память и обратно включалось в состав измеренного времени выполнения фазы 1 и 2 соответственно.

Измерено время пересечения таких пар многоугольников, в которых 2-й полигон получен смещением 1-го на вектор. В измерениях п. *А* этот вектор мал, чтобы было много пересечений рёбер; в измерениях п. *В* для одних и тех же полигонов этот вектор варьируется.

А. Исследование зависимости времён выполнения от количества вершин при большом числе пересечений рёбер.

Количество пересечений рёбер примерно равно половине суммарного количества вершин. Результаты измерений последовательной и параллельной версий алгоритма для разного количества вершин приведены в табл. 2.

Из таблицы можно сделать следующие выводы:

– параллельный алгоритм работает быстрее последовательного уже начиная с порядка 1000 вершин;

– в данном случае 1-я фаза отнимает около 94 % времени в последовательном алгоритме и около 96 % времени в параллельном;

– параллельный алгоритм в целом достигает ускорения в 12,7 раза, по фазе 1 – в 12,3 раза, а по фазе 2 – в 24 раза.

В. Исследование зависимости времён выполнения от количества пересечений рёбер при большом числе вершин.

Использован полигон с 33 378 вершинами. Из результатов измерений времён выполнения последовательной и параллельной версий алгоритма для разного количества пересечений рёбер можно сделать следующие выводы:

– число пересечений практически не влияет на длительность фазы 1 в последовательном алгоритме и влияет в параллельном алгоритме;

– число пересечений сильно влияет на длительность фазы 2;

– несмотря на то, что фаза 2 распараллелена лучше фазы 1 (ускорение фазы 2 – до 32 раз), параллельный алгоритм оказывается сравнительно эффективнее при меньшем числе пересечений (достигает ускорения в 13 раз), так как при этом уменьшается длительность его фазы 1 (она достигает ускорения в 13 раз) (неизвестно, сохранится ли эта тенденция при увеличении числа пересечений до порядка квадрата от числа вершин);

– длительность фазы 2 линейно зависит от числа пересечений, как и должно быть.

Разработанный модифицированный расширенный алгоритм Грейнера–Хорманна для CUDA демонстрирует значительное ускорение над своим последовательным аналогом (до 13 раз) и может быть использован (после простых модификаций) при решении задач, требующих пересечения простых полигонов с большим количеством вершин.

В дальнейшем представляется целесообразным: оптимизация алгоритма с учётом более тонких особенностей CUDA; исследование эффективности других вариантов реализации модифицированного расширенного алгоритма Грейнера–Хорманна для CUDA (исследована эффективность лишь одного из возможных вариантов реализации), в том числе использования эффективных алгоритмов поиска всех пересечений рёбер; обработка особых случаев в модифицированном расширенном алгоритме Грейнера–Хорманна, связанных с ограниченной точностью чисел типа double; дальнейшие исследования влияния характеристик входных данных на время работы вариантов модифицированного расширенного алгоритма Грейнера–Хорманна на CUDA.

Таблица 2

Количество вершин в 1-м полигоне	Число пересечений	Последовательный алгоритм (мс)			Алгоритм для CUDA (мс)		
		Фаза 1	Фаза 2	Фазы 1 и 2	Фаза 1	Фаза 2	Фазы 1 и 2
340	347	0,005	0,000	0,005	0,023	0,001	0,024
564	575	0,014	0,001	0,015	0,025	0,001	0,026
1176	1110	0,058	0,003	0,061	0,035	0,002	0,037
1828	1825	0,140	0,009	0,149	0,036	0,002	0,038
3474	3350	0,50	0,03	0,53	0,066	0,004	0,070
6120	6163	1,55	0,11	1,66	0,151	0,008	0,159
10132	10092	4,2	0,3	4,5	0,38	0,02	0,40
18224	18222	13,8	0,9	14,7	1,12	0,04	1,16
33378	34513	45,4	3,3	48,7	3,77	0,14	3,91
55372	57218	–	–	–	10,0	0,4	10,4

СПИСОК ЛИТЕРАТУРЫ

1. Скворцов А. В. Построение объединения, пересечения и разности произвольных многоугольников в среднем за линейное время с помощью триангуляции // Вычислительные методы и программирование. 2002. Т. 3. С. 116–123.

2. Ченцов О. В., Скворцов А. В. Обзор алгоритмов построения оверлеев многоугольников // Вестн. Томского гос. ун-та. 2003. № 280. С. 338–345.

3. Dae Hyun Kim. An Extension of Polygon Clipping To Resolve Degenerate Cases // Computer-Aided Design & Applications. 2006. Vol. 3, № 1–4. P. 447–456.

4. Jianting Zhang. Polygon Overlay Operations in CudaGIS (Initial Design & Implementation) / Jianting Zhang, Simin You, Kai Zhao. – Режим доступа: http://www-cs.ccny.cuny.edu/~jzhang/papers/PolyOvr_dr.pdf

5. Cederman D., Tsigas Ph.GPU-Quicksort: A Practical Quicksort Algorithm for Graphics Processors // ACM J. of Experimental Algorithmics (JEA). 2009. Vol. 14, Dec (секция 1, статья № 4).

6. Препарата Ф., Шеймос М. Вычислительная геометрия. М.: Мир, 1989. (См. также: Point in polygon // Wikipedia. – Режим доступа: http://en.wikipedia.org/wiki/Point_in_polygon#Ray_casting_algorithm.)

M. A. Firsov, S. A. Ivanovskiy

PARALLEL IMPLEMENTATION OF ALGORITHM FOR CONSTRUCTION OF SIMPLE PLANE POLYGON INTERSECTION WITH USE OF CUDA TECHNOLOGY

Parallel implementation of modified extended Greiner-Hormann algorithm for construction of simple plane polygon intersection with use of CUDA is described. Experimental comparison results of sequential version and version for CUDA run times are given.

Computational geometry, simple polygon intersection, CUDA