Алгоритм восстановления поверхности из облака точек на графическом процессоре

Дмитрий Козлов
Нижегородский Государственный
Университет им. Н.И. Лобачевского
dmitry.a.kozlov@gmail.com

Вадим Турлапов
Нижегородский Государственный
Университет им. Н.И. Лобачевского
vadim.turlapov@cs.vmk.unn.ru

Аннотация

В данной работе представлен алгоритм восстановления поверхности из облака точек трехмерного пространства, расположенных вблизи некоторой двусторонней замкнутой поверхности M. Описанный алгоритм принимает на входе множество точек $\{x_1, \dots, x_n\} \subset \mathbb{R}^3$ и порождает на выходе триангуляцию, аппроксимирующую поверхность M. Алгоритм не использует никакой дополнительной информации, кроме содержащейся в наборе входных точек, извлекая информацию о границах поверхности и ее топологии из особенностей входных данных. Алгоритм разработан для выполенения на устройствах с массовым параллелизмом и реализован при помощи технологии гетерогенных вычислений OpenCL. Приложениями вышеуказанного алгоритма могут выступать обработка данных с 3D сканера, восстановление объектов по набору сечений, обработка моделей с точечным представлением и т.д.

Ключевые слова

Восстановление поверхностей, анализ данных со сканера, графический процессор, OpenCL, CUDA, анализ главных компонент, метод марширующих кубов.

1. ВВЕДЕНИЕ

Проблема восстановления поверхности из облака точек встает во множестве областей компьютерной графики, одной из которых является анализ данных, полученных с 3D сканера. Такие данные обычно представляются двумерным массивом значений расстояния от сенсора сканера до соответствующей точки объекта (карта глубины). Сканирование может выполняться с нескольких точек обзора и порождать множество таких наборов данных. Так же облака точек могут быть получены при помощи различного медицинского оборудования и программного обеспечения. Первым этапом анализа данных такого типа обычно является восстановление поверхности и ее триангуляция с целью получить базовую аппроксимацию. Триангуляция может использоваться в дальнейшем для параметризации такой поверхности функциями более высоких порядков. В данной области было разработано множество алгоритмов, часть из которых использует априорные данные о входных точках[6], другая же опирается только на информацию, содержащуюся во входном наборе [5,2]. Отличительной чертой всех алгоритмов восстановления поверхностей является их исключительная ресурсоемкость, обусловленная необходимостью обработки большого числа однотипных данных. В основе работы лежит алгоритм, приведенный в статье [5], который был существенным образом переработан для эффективного исполнения на архитектурах с массовым параллелизмом, разработаны структуры хранения данных, рассчитанные на такие устройства.

2. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В области восстановления поверхностей было предложено множество алгоритмов, которые можно классифицировать по различным признакам. Как уже говорилось алгоритмы восстановления можно разделить по использованию априорных данных на те, которые используют дополнительные знания о входных точках и те, которые опираются только на информацию, заключенную во входном наборе. Работа [2], например, использует гладкость многообразия, в то время как алгоритм, описанный в [6] не предполагает каких-либо «хороших» свойств у поверхности M. Алгоритмы восстановления так же могут быть классифицированы по типу функции, аппроксимирующей поверхность. В качестве такой функции может использоваться неявное или параметрическое представление. В работах [6,10] строится аппроксимация поверхности суммой Гауссовых функций с различными параметрами, а в [15,16] используется построение параметрической модели над прямоугольной областью определения. Методы «подгонки» параметров функции тоже сильно разнятся. В работе [14] используются вариации метода наименьших квадратов с различными вариантами метрик. Работа [13] опирается на метод деформации сфер, а [2] построена на базе диаграмм Вороного и так называемых α-форм. До недавнего времени предпринималось сравнительно мало попыток распараллеливания алгоритмов восстановления поверхностей, из которых стоит отметить [8], где используется октодерево в качестве структуры данных для хранения обрабатываемых точек. В работе [6] исследуется масштабируемость алгоритма восстановления на основе поверхностей Пуассона при использовании адаптивной сетки в качестве ускоряющей структуры.

3. АЛГОРИТМ

Приведем краткий обзор используемого алгоритма. Алгоритм состоит из двух основных стадий. На первой происходит аппроксимация неявной функции $F: D \to \mathbb{R}$, где $D \subset \mathbb{R}^3$ некоторый слой, заключающий в себе поверхность M. Данная функция представляет собой оценку расстояния от любой точки слоя D до поверхности M. Ее значение может быть положительным или отрицательным, в зависимости от того с какой стороны поверхности находится точка. На второй стадии алгоритма производится триангуляция поверхности

нулевого уровня функции F при помощи алгоритма марширующих кубов [9].

3.1 Восстановление поля нормалей

Для построения функции расстояния мы, следуя [5] ассоциируем с каждой точкой набора $\{x_1, \dots, x_n\}$ ориентированную касательную плоскость $Tp(x_i)$. Каждая такая плоскость описывается некоторой точкой o_i и нормалью n_i. Точка o_i рассчитывается как центр масс k ближайших к x_i точек:

$$o_i = \frac{\sum_{x \in KNbh(x_i)} x}{K}$$

Нормаль n_i определяется, используя те же ближайшие k точек при помощи метода главных компонент. Для этого нужно построить ковариационную матрицу для данного набора из k точек:

$$CV = \sum_{x \in KNbh(x_i)} (x - o_i) \otimes (x - o_i)$$

Где $\otimes$ обозначает тенезорное произведение векторов. Ковариационная матрица является симметричной и положительно-полуопределенной. Далее необходимо найти собственный вектор этой матрицы, соответствующий минимальному по модулю собственному числу. Данный вектор характеризует направление, в котором разброс точек минимален. Это направление мы и принимаем за приближенное направление нормали.

3.2 Ориентация нормалей

Если две точки x_i и x_j расположены достаточно близко, то их нормали должны быть практически параллельными ($n_i n_j \approx 1$). Тем не менее, алгоритм их нахождения, описанный выше, не гарантирует нам такой ориентации, они могут быть направлены в противоположные стороны. Таким образом, в алгоритме возникает подзадача корректной ориентации нормалей. Стоит заметить, что для некоторых приложений алгоритма, таких как задача обработки сканированных данных, этот шаг алгоритма может быть опущен, так как данные с 3D сканера уже содержат, как правило, направление нормали к поверхности. Для решения данной задачи строится граф $G = (V, E)$, где множеству вершин соответствуют точки данных, а ребро $(v_i, v_j) \in E$ если x_i является одной из k ближайших к x_j точек. Далее каждому ребру (v_i, v_j) присваивается вес $1 - |n_i n_j|$, после чего задача сводится к нахождению минимального остовного дерева графа G и распространению вдоль его ребер правильной ориентации, что соответствует распространению по направлению минимальной кривизны. За начальную ориентацию нормали выбирается положительное направление оси z для точки с максимальной z координатой, после чего производится распространение направления от данной вершины по ребрам минимального остовного дерева. В нашей реализации используется параллельный алгоритм Борувки для построения минимального остовного дерева, описанный в [3].

3.3 Вычисление функции расстояния

Функция расстояния является оценкой расстояния от точки слоя D до поверхности M. Ее вычисление осуществляется по следующей формуле:

$$F(p) = (p - o_i) n_i$$

Где o_i – ближайший к p «центр» касательной плоскости, а n_i– соотвестсвующая ему нормаль.

3.4 Триангуляция

Триангуляция осуществляется при помощи известного алгоритма марширующих кубов [9], предназначенного для триангуляции поверхностей равного уровня скалярного поля. Данный алгоритм основан на вычислении значений упомянутого скалярного поля в узлах регулярной сетки и последующей генерации треугольников для каждой из ячеек сетки на основе битовой маски. Мы используем адаптивную модификацию этого алгоритма для построения триангуляции множества $F(p) = 0$, которое является аппроксимацией поверхности M.

4. ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ

4.1 Терминология

В нашей параллельной реализации широко используются примитивы параллельного программирования, описанные в [3,4]. Таблица 1 содержит краткое описание таких примитивов.

Название операции	Краткое описание	Сложность (CRCW PRAM)
Reduce	Редукция массива с использованием ассоциативной операции	$O(\log_2 N)$
Scan	Нахождение префиксной суммы	$O(\log_2 N)$
Compact	Удаление повторяющихся элементов	$O(\log_2 N)$
Sort	Сортировка массива	$O((\log_2 N)^2)$

Таблица 1. Параллельные примитивы

4.2 Ускоряющая структура

Для эффективного исполнения алгоритма на GPU необходимо применение ускоряющей структуры для входного набора данных. Используемые нами алгоритмы предъявляют следующие требования к ускоряющей структуре:

- Эффективный доступ к информации о ближайших соседях точки
- Эффективный алгоритм построения структуры на GPU (в случае динамического облака)
- Эффективное использование видео памяти
- Простота реализации

Основываясь на этих требованиях, мы выбрали в качестве ускоряющей структуры регулярную сетку. Был разработан эффективный алгоритм параллельного построения данной структуры и удобный формат хранения данных, позволяющий избежать избыточного использования видео памяти. Листинг 1 содержит описание алгоритма формирования регулярной сетки в терминах приведенных ранее параллельных примитивов.

Листинг 1. Параллельное построение регулярной сетки

```
// Шаг 1: Построение bounding box'a
1: BBox = new bbox
2:Reduce(Q,BBox, union operator)
// Шаг 2: Вычисление идентификаторов точек
3: Pid = new array [N]
4: for each i = 0 to N-1 in parallel
5:          pid[i] = VoxelId(Q[i]) << 32 + i
// Шаг 3: Сортировка идентификаторов точек
6: SortedPid = new array [N]
7: Sort(Pid,SortedPid,N)
// Шаг 4: Переупорядочивание точек
8: SQ  = new array[N]
9: for each i = 0 to N-1 in parallel
10:         SQ[i] = Q[SortedPid[i] & 0xFFFFFFFF]
// Шаг 5: Выделение вокселей сетки
11:Marks = new array [ N + 1]
12: for each i = 1 to N-1 in parallel
13:         if ( (SortedPid[i]  >> 32) != (SortedPid[i-1]>>32)
14:                 Marks[i] = 1
15:         else Marks[i] = 0
16: Scan(Marks, Marks, +)
17: NumVoxels = Marks[N]
18: Voxels = new array [NumVoxels]
19: Compact(Voxels, Marks, SQ)
```

На первом шаге алгоритма строится bounding box облака точек, путем применения примитива Reduce к исходному массиву точек. Второй шаг предназначен для вычисления уникальных 64-битных идентификаторов точек, которые позволят переупорядочить массив точек таким образом, что точки, принадлежащие одному и тому же вокселю, будут располагаться в видеопамяти последовательно. На третьем шаге производится сортировка массива идентификаторов. Первичным ключом сортировки является идентификатор вокселя, вычисляемый как индекс в развертке трехмерного массива, вторичным – индекс точки в начальном массиве. Далее массив точек переупорядочивается в соответствии с новым порядком идентификаторов. Таким образом, точки, попадающие в один воксель сетки, лежат в видеопамяти последовательно, что позволяет сохранять лишь индекс начальной точки и количество точек, используя константный объем памяти для каждого вокселя. На пятом шаге происходит выделение групп точек, принадлежащих одному и тому же вокселю. Для этого строится массив маркеров в котором i-й элемент содержит единицу если точки с индексами i и i-1 принадлежат различным вокселям и ноль в противном случае. Путем применения примитива Scan к массиву маркеров мы находим для каждой точки индекс первой точки текущего вокселя. Примитив Compact для каждого вокселя сохраняет индекс первой точки в массиве и количество точек, содержащихся в данном вокселе. Стоит отметить, что в памяти сохраняются только те воксели, которые содержат точки данных, что позволяет значительно снизить использование видеопамяти.

4.3 Восстановление поля нормалей

Для построения оценки расстояния необходимо нахождение касательной плоскости для каждой из точек входных данных. Далее приводится описание параллельного алгоритма построения оценки расстояния с использованием нашей ускоряющей структуры.

1. Получить из глобальной памяти обрабатываемую точку
2. Определить целочисленные координаты вокселя (x,y,z)
3. Выполнить выборку индексов точек вокселя (x,y,z) и его соседей
4. Скопировать точки с полученными индексами в локальную память
5. Отсортировать точки по возрастанию расстояния
6. Выбрать из локальной памяти k ближайших точек
7. Вычислить центроид c и поместить его в выходной буфер
8. Вычислить матрицу ковариации для выбранных точек
9. Найти минимальное собственное число и поместить соответствующий ему собственный вектор в выходной буфер

4.4 Вычисление оценки расстояния

Для вычисления оценки расстояния мы помещаем центроиды в ускоряющую структуру, что позволяет использовать следующую параллельную схему:

1. Получить из глобальной памяти обрабатываемую точку
2. Определить целочисленные координаты вокселя (x,y,z)
3. Найти ближайший центроид и соответствующую ему нормаль, используя ускоряющую структуру
4. Вычислить оценку расстояния

4.5 Алгоритм марширующих кубов

Нахождение поверхности нулевого уровня оценки расстояния осуществляется при помощи алгоритма марширующих кубов, который состоит в расчете значения функции в узлах регулярной сетки и последующей генерации треугольников для каждой ячейки этой сетки. Благодаря разработанной нами структуре, алгоритм существенным образом ускоряется, за счет того, что им обрабатываются только те воксели, которые действительно могут содержать поверхность нулевого уровня оценки расстояния. В листинге 2 приведено описание алгоритма.

Листинг 2. Алгоритм марширующих кубов

```
// Шаг 1: Вычисление шаблонов ячеек
1: Case = new array [NumVoxels]
2: for each i = 0 to NumVoxels-1 in parallel
3:         Case[i] = ComputeCase(F, Voxels[i]);
// Шаг 2: Вычисление адресов вершинного буфера
4: NumTris = new array [NumVoxels + 1]
5: for each i = 0 to NumVoxels-1 in parallel
6:         NumTris[i] = LookUp(Case[i]);
7: Scan(NumTris, NumTris, + )
8: TotalTris = NumTris[NumVoxels]
// Шаг 3: Генерация треугольников
8:Tris = new array [TotalTris]
9: for each i = 0 to NumVoxels-1 in parallel
10:        GenerateTris(Tris, NumTris[i]);
```

На первом шаге алгоритма для каждого вокселя вычисляется маска, которая определяет конфигурацию треугольников в данном вокселе. Для того, чтобы создать и заполнить вершинный буфер необходимо определить точное количество треугольников, генерируемое для всех вокселей. На втором шаге создается массив, каждый элемент которого содержит количество треугольников, содержащееся в соответствующем вокселе. Применяя примитив Scan к данному массиву, мы получаем общее количество треугольников и индексы для записи треугольников для каждого вокселя. Такая техника позволяет получить выходные данные различной длины от каждого потока в один буфер параллельно.

5. ОГРАНИЧЕНИЯ АЛГОРИТМА

Алгоритм применяется для восстановления замкнутых двусторонних поверхностей, но согласно [5], может быть модифицирован для восстановления поверхностей с границами. Предположим, что набор входных точек имеет плотность ρ (это означает, что любая сфера с центром, лежащим на M радиуса ρ содержит, по крайней мере, одну точку входного набора) и максимальный уровень шума δ (т.е. положение точки входных данных может отклоняться от поверхности не более, чем на δ. Очевидно, что если $d(p) > \rho + \delta$, где $d(p)$ - проекция точки на ближайшую касательную плоскость, то такая точка не может лежать на поверхности M. Таким образом, мы приходим к более общему правилу вычисления $F(p)$.

$$F(p) = \begin{cases} (p - o_i)n_i \text{ если } o_i - \big((p - o_i)n_i\big)n_i \le \rho + \delta \\ \text{не определено в противном случае} \end{cases}$$

6. ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ

В таблице 2 приведены результаты измерения производительности алгоритма для сетки размером 256x256x256. Для тестирования использовалась рабочая станция с процессором Intel Core i7 975 Extreme 3,33Ггц и видеоадаптером ATI Radeon HD5870 с 1Гб видеопамяти.

Модель	Кол-во точек	Кол-во обработанных ячеек сетки	Время, мс			
			Построение сетки	Построение F	Триангуляция	Общее
Sphere	65536	52502	12	6	3	21
Sphere	131072	90208	26	11	11	48
Bunny	353272	178714	40	140	20	200

Таблица 2. Производительность

7. ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан параллельный алгоритм восстановления поверхности из облака точек для архитектур с массовым параллелизмом. Были разработаны и реализованы эффективные структуры хранения данных, а так же предложена архитектура системы для реализации в рамках технологии гетерогенных вычислений OpenCL. В дальнейшем работу предполагается использовать в качестве основы для построения алгоритмов аппроксимации восстановленной поверхности функциями более высоких порядков. Полученные показатели времени работы свидетельствуют о том, что алгоритм может быть использован для восстановления поверхности в реальном времени.

8. ССЫЛКИ

[1] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., And Silva, C. T. 2001. Point set surfaces. In Proceedings of IEEE Visualization'01, 21–28.

[2] Amenta, N., Bern, M., and Kamvysselis, M. 1998. A new Voronoi-based surface reconstruction algorithm. In Proceedings of SIG-GRAPH'98, 415–421.

[3] Blelloch, G. Vector Models for Data-Parallel Computing. 1990. The MIT Press

[4] Harris, M., Sengupta, S., and Owens, J. D. 2007. Parallel prefix sum (scan) with CUDA. In GPU Gems 3, H. Nguyen, Ed. Addison Wesley, Upper Saddle River.

[5] Hoppe, H., Derose, T., Duchamp, T., Mcdonald, J., and Stuetzle, W. 1992. Surface reconstruction from unorganized points. In Proceedings of SIGGRAPH'92, 71–78.

[6] Kazhdan, M., Bolitho, M., and Hoppe, H. 2006. Poisson surface reconstruction. In Proceedings of SGP'06, 61–70

[7] Kruskal, J., On the shortest spanning subtree of a graph and the traveling salesman problem. // Proc. AMS. 1956. Vol 7, No. 1. C. 48–50

[8] Kun, K., Minmin, G., Xin, H., Baining, G., Highly parallel surface reconstruction. Microsoft research Asia.

[9] Lorensen, W. E., And Cline, H. E. 1987. Marching cubes: A highresolution 3d surface construction algorithm. In Proceedings of SIG-GRAPH'87, 163–169.

[10] Muraki, S. Volumetric shape description of range datausing "blobby model". *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):227–235, July 1991.

[11] Ohtake, Y., Belyaev, A., Alexa, M., Turk, G., And Seidel, H.-P.2004. Multi-level partition of unity implicits. ACM Trans. Graph. 22, 3, 463–470.

[12] Smith, K., Eigenvalues of a symmetric 3x3 matrix. Commun. ACM 4(4): 168 (1961))

[13] Taubin, G. Estimation of planar curves, surfaces and nonplanar space curves defined by implicit equations, with applications to edge and range image segmentation. Technical Report LEMS-66, Division of Engineering, Brown University, 1990

[14] Turk, G., and O'Brien, J. F. 2002. Modelling with implicit surfaces that interpolate. ACM Trans. Graph. 21, 4, 855–873.

[15] Vemuri, B. *Representation and Recognition of ObjectsFrom Dense Range Maps*. PhD thesis, Department of Electrical and Computer Engineering, University of Texas at Austin, 1987.

[16] Vemuri, B., Mitiche, A., and Aggarwal, J. Curvaturebased representation of objects from range data. *Image and Vision Computing*, 4(2):107–114, 1986.

[17] Yusov, E., Turlapov, V. 2008. JPEG2000-based compressed multiresolution model for real-time large scale terrain visualization. В кн.: Conference Proceedings of the 18th international Conference on Computer Graphics and Vision "GraphiCon'2007", 2007. С. 164—171

9. РЕЗУЛЬТАТЫ РАБОТЫ АЛГОРИТМА

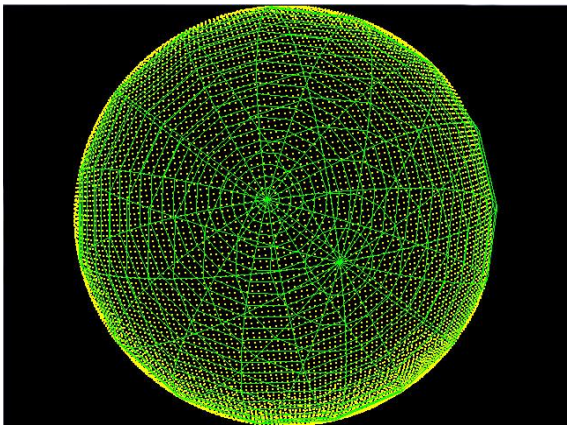

Рисунок 1. Облако точек на сфере

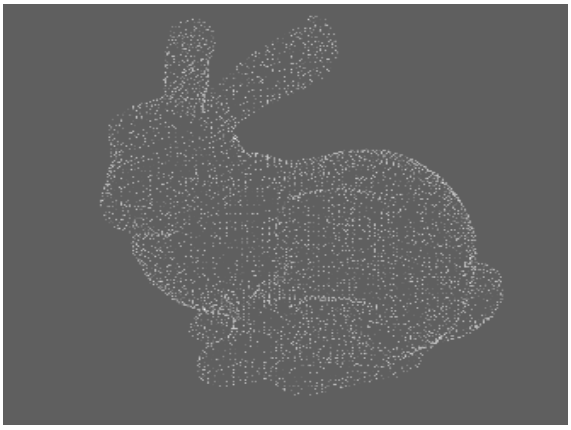

Рисунок 3. Облако точек (Стэнфордский кролик)

-

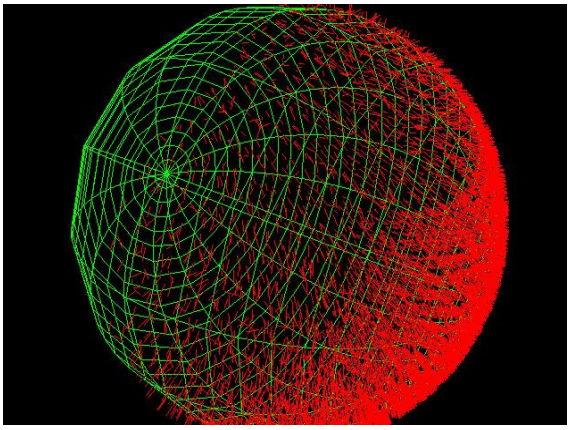

Рисунок 2. Восстановленное поле нормалей

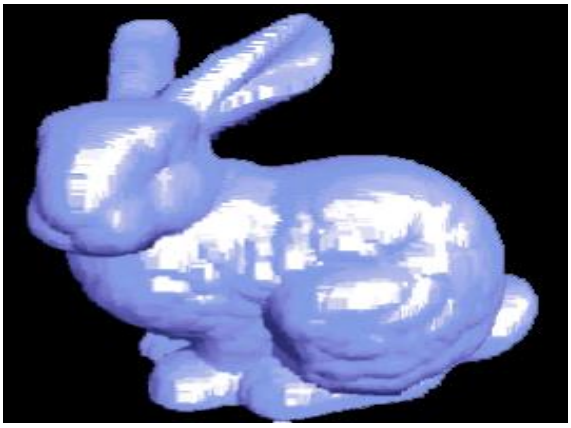

Рисунок 4. Тригуляция с освещением