Параллельная реализация алгоритма сегментации разноракурсных изображений с использованием преобразования Хафа

Е.В. Гошин[а,б], А.П. Котов[а], Г.Е. Лошкарева[а]

[а] *Самарский национальный исследовательский университет имени академика С.П. Королёва, 443086, Московское шоссе, 34, Самара, Россия*
[б] *Институт систем обработки изображений РАН – филиал ФНИЦ «Кристаллографияи фотоника» РАН, 443001, ул. Молодогвардейская, 151, Самара, Россия*

Аннотация

Разработана параллельная реализация алгоритма сегментации разноракурсных изображений посредством сегментации соответствующей трёхмерной сцены. Алгоритм включает в себя формирование трёхмерной модели сцены в виде облака точек и сегментацию полученного облака точек в трёхмерном пространстве с использованием пространства Хафа. Разработанный параллельный алгоритм реализован на графических процессорах с использованием технологии CUDA. Проведены эксперименты для оценки ускорения и эффективности предлагаемого алгоритма. Разработанная параллельная программа протестирована на модельных сценах.

Ключевые слова: сегментация; трёхмерная модель; преобразование Хафа; CUDA

1. Введение

Анализ, обработка изображений, а также задача сегментации изображений является наиболее востребованной в различных сферах деятельности человека. Многие современные приложения основываются на обработке информации. Часто этой информацией являются снимки с камер. Однако, не редко в каких-то изображениях крайне мало необходимой информации для того, что сегментации соответствовала хоть в какой-то степени реальным фактам. Например, такое может произойти в случаях, когда текстура объектов сцены состоит из областей разных цветов достаточно крупного размера. В данном случае, если есть некоторое количество изображений, имеет смысл рассматривать и изучать не яркостную характеристику отдельных снимков, а визуализированную трёхмерную структуру сцены.

Существует значительное число алгоритмов и методов для того, что сформировать по разноракурсным изображением трехмерную модель сцены [2, 3]. Часто возникает такая ситуация, когда в случае изображений, которые получены с различных ракурсов, не имеющих какой-либо зависимости, в таком случае получаем, что параметры съёмки неизвестны и возникает вопрос в определения этих параметров. Этой задаче, а именно, сопоставления разноракурсных изображений в такой ситуации посвящен ряд статей [4, 5].

В сегментации трех сцен можно выделить такой подход как выделение объектов на этой сцене, имеющие некоторый сходный характер [6, 7]. Например, в работе [8] рассматривается случай выделения плоскостей по облаку точек в сцене, т.е. ее сегментации (облако точек, в данном случае, было получено с помощью сканирования поверхности Земли лидаром). В нашем же случае, для выделения плоскостей будет использовано трехмерное преобразование Хафа.

Целью настоящей работы является ускорение технологии, предложенной в работе [9] за счёт параллельной реализации одного из этапов этой технологии: поиска наиболее подходящих плоскостей при помощи пространства Хафа. В завершении работы приведены экспериментальные исследования ускорения параллельной реализации алгоритма по сравнению с последовательной реализацией.

2. Описание технологии

Основные этапы рассматриваемой технологии сегментации разноракурсных изображений с использованием трёхмерного преобразования Хафа, предложенной в работе [9], приведены в виде общей схемы на рисунке 1.

Согласно схеме, вначале строится трехмерная модель сцены по двум изображениям. Затем над всеми точками полученной трёхмерной сцены проводится преобразование Хафа. Среди всех найденных плоскостей с помощью преобразования ищется максимум в аккумуляторном пространстве, посредством чего выбирается «базовая» плоскость. Далее, выполняя расчет расстояния от точек до выбранной плоскости, из одной модели получаем две, на одной из которых отображены точки фона, на другой же – точки объектов. После всех этих шагов по полученной (сегментированной) сцене по необходимости можно сегментировать исходное изображение. Далее будут рассмотрены ключевые этапы технологии.

В настоящей работе используется алгоритм определения параметров съёмки, описанный в работе [9]. Предполагается, что матрицы первой и второй камер одинаковы и известны, глобальная система координат совпадает с системой координат первой камеры, т. е. направление съемки совпадает с направлением оси *OZ*, и на паре изображений задано множество соответствующих точек $\left\{\mathbf{m}_i \mid i=\overline{1,N}\right\}$ и $\left\{\mathbf{m}'_i \mid i=\overline{1,N}\right\}$, где точки $\mathbf{m}_i=\begin{pmatrix}x & y\end{pmatrix}^T$ и $\mathbf{m}'_i=\begin{pmatrix}x' & y'\end{pmatrix}^T$ каждой пары являются проекциями одной точки в трёхмерном пространстве.

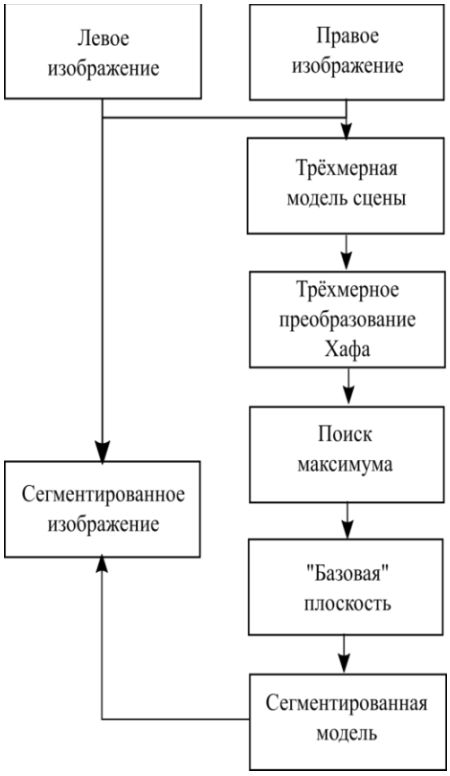

Рис. 1. Схема технологии.

В ходе результате работы алгоритма определяется матрица поворота $\mathbf{R}'$ и вектор сдвига $\mathbf{t}' = \begin{pmatrix} t'_x & t'_y & t'_z \end{pmatrix}$ второй камеры относительно первой.

С использованием алгоритма Лукаса-Канаде [10] формируется оптический поток, устанавливающий соответствие между точками первого и второго изображений. Посредством пространственной триангуляции [11] на основе полученных соответствий формируется облако точек.

Целью этапа сегментации трёхмерной модели сцены является отделение объектов от фона на сцене.

Для выделения плоскостей (фон и объекты на сцене) применяется трёхмерное преобразование Хафа. Преобразование Хафа – это способ поиска параметрических объектов, который, чаще всего, используется для отыскания линий и кругов, и других форм на изображении. Например, в статье [13] описано как использовать обобщенный метод Хафа для детектирование различных двумерных объектов по эталонному контуру.

При преобразовании Хафа для всех заданных точек в исходном пространстве делается предположение о том, принадлежат они искомому объекту или нет. Таким образом, для этого находится и решается уравнение для каждой точки изображения (в нашем случае сцены), получаются некоторые параметры, которые составляют пространство Хафа. Финальным шагом мы идем по всему пространству Хафа и выбираем максимальные значения. Максимальным значением будет то, за которое было отдано большее количество голосов координатами изображения (сцены). Отсюда получаем и параметры для уравнения искомого нами объекта, будь то, линия, круг или какая-то другая фигура.

Существуют также множество модификаций преобразования Хафа: вероятностное, случайное, иерархическое, размытие фазового пространства, использование градиента яркости изображения и другие.

В качестве входного и выходного значений параметризованной плоскости будем использовать набор точек их трехмерного вещественного пространства. Плоскость представим как вектор нормали к этой плоскости и расстояние от начала координат до плоскости. На плоскости пусть дана точка $\mathbf{p}$ на плоскости, $\mathbf{n}$ – вектор нормали, который перпендикулярен плоскости, расстояние ρ. Таким образом, расстояние ρ будет выглядеть следующим образом:

$$\rho = \mathbf{p} \cdot \mathbf{n} = p_x n_x + p_y n_y + p_z n_z .$$

После подстановки выражений для углов между вектором нормали и выбранной системой координат уравнение плоскости приобретает вид:

$$p_x \cdot \cos\theta \cdot \sin\varphi + p_y \cdot \sin\varphi \cdot \sin\theta + p_z \cdot \cos\varphi = \rho , \quad (1)$$

где θ, φ – углы, задающие вектор нормали. Координаты φ, θ и ρ составляют такое трёхмерное пространство Хафа, что каждой его точке соответствует одна плоскость в вещественном трехмерном пространстве. В свою очередь каждой точке (x_0, y_0, z_0) вещественного трехмерного пространства соответствует в пространстве Хафа некоторая поверхность, при этом каждая точка этой поверхности (φ, θ, ρ) характеризует некоторую плоскость, проходящую через искомую точку (x_0, y_0, z_0).

В данной работе решается задача поиска «базовой» плоскости из сформированного облака точек, которая содержит наибольшее число точек. Для всех точек из исходного облака после определения параметров $(\hat{\varphi}, \hat{\theta}, \hat{\rho})$ «базовой» плоскости определяется, принадлежит ли эта точка плоскости или нет. Чтобы это выяснить координаты точки

подставляются в уравнение плоскости. Далее мы получаем некоторое значение, которое сравниваем с некоторым порогом:

$$p_x \cdot \cos\hat{\theta} \cdot \sin\hat{\varphi} + p_y \cdot \sin\hat{\varphi} \cdot \sin\hat{\theta} + p_z \cdot \cos\hat{\varphi} - \rho < \Delta . \quad (2)$$

Все точки относятся к плоскости, если они удовлетворяют этому неравенству, остальные – считаются объектами сцены.

Результаты сегментации модели могут быть использованы для сегментации исходных изображений, поскольку существует взаимно однозначное соответствие между пикселями изображений и восстановленными точками трёхмерной модели.

3. Последовательная реализация алгоритма трехмерного преобразования Хафа

Рассмотрим алгоритм, который используется для реализации трехмерного преобразования Хафа в настоящей работе. Трехмерный массив целочисленных значений используется в качестве аккумуляторного пространства. Каждой ячейке в этом пространстве соответствует плоскость с параметрами, которые задаются с помощью использования координат этой ячейки.

Поскольку точного соответствия добиться невозможно вследствие дискретности отсчётов массива, то из сформированного на предыдущем этапе облака точек, каждая точка увеличивает на единицу значение тех ячеек аккумуляторного массива, которые соответствуют плоскостям, проходящим через данную точку или вблизи её.

С использованием псевдокода приведенный алгоритм запишем следующим образом:

Последовательная реализация

Начальные данные: Облако точек
Результирующие данные: Аккумуляторный массив

Цикл по всем точкам облака данных (x_0, y_0, z_0)
  Цикл по всем углам θ от 0 до π с шагом $\pi/180$
    Цикл по всем углам φ от 0 до π с шагом $\pi/360$
      Вычисление ρ: $\rho = p_x \cdot \cos\theta \cdot \sin\varphi + p_y \cdot \sin\varphi \cdot \sin\theta + p_z \cdot \cos\varphi$
      Приведение к целому типу ρ
      Если $\rho < \Delta$
        Операция инкремента: $A(\theta, \phi, \rho) = A(\theta, \phi, \rho) + 1$
    Конец цикла по углу φ
  Конец цикла по углу θ
Конец цикла по точкам облака данных
Поиск максимального элемента $A(\theta, \phi, \rho)$

В результате после выполнения этого алгоритма каждой ячейке полученного массива будет поставлено в соответствие число, определяемое как количество точек из исходного облака точек, лежащих вблизи плоскости, задаваемой этой ячейкой. В результате та ячейка массива, которая будет иметь максимальное значение, и будет искомой точкой, задающей «базовую» плоскость.

4. Параллельная реализация предложенного алгоритма

Преобразование Хафа и, в частности, предложенный алгоритм являются вычислительно затратными по времени, вследствие нерегулярного доступа к памяти при операции инкремента аккумуляторного массива. Использование технологии CUDA (Compute Unified Device Architecture) позволяет декомпозировать данную операцию. Однако, из-за вышеупомянутого нерегулярного и непредсказуемого доступа к памяти, эффективная реализация алгоритма преобразование Хафа на графическом процессоре не тривиальна [14].

Архитектура NVIDIA GPU (Graphical Processing Unit) основывается на потоковых мультипроцессорах (streaming multiprocessors, SMs), масштабируемых по числу нитей (threads). Каждый мультипроцессор GPU запускает выполнение тысячей нитей одновременно. Когда CUDA программа на CPU (host CPU) вызывает сетку ядер GPU (kernel grid), блоки (thread block), из которых состоит сетка, распределяются между потоковыми мультипроцессорами (SMs). Ядром GPU (kernel) принято называть часть кода CUDA программы, выполняемого на GPU. Нити выполняют одинаковую программу (ядро GPU) не обязательно одновременно. Одновременно выполняются нити, объединённые в одном блоке нитей. Нити внутри блока нитей расположены в варпах (warp), по 32 нити в одном варпе. Каждая нить в варпе за один такт выполняет одинаковую инструкцию [15].

Предложенный алгоритм трехмерного преобразования Хафа реализован в виде CUDA программы. В CUDA программе часть кода исполняется либо на CPU (host), либо на GPU (device). Алгоритм работы реализованной

программы состоит из пяти последовательно выполняющихся этапов. Ниже приведены этапы. В скобках указывается устройство, выполняющее процедуры на данном шаге: host или device.

Основные этапы CUDA программы:

1. Выделение памяти для входных и результирующих данных в глобальной памяти GPU (host);
2. Копирование входных данных из оперативной памяти в глобальную память GPU (host);
3. Выполнение сетки ядер GPU (kernel grid) и сохранение вычисленных значений аккумуляторного массива в глобальной памяти GPU (device);
4. Копирование полученных результатов из глобальной памяти GPU в оперативную память (host).
5. Освобождение глобальной памяти (host).

После формирования аккумуляторного массива нахождение параметров искомой плоскости является тривиальной.

Для вышеуказанной схемы CUDA программы было рассмотрено две реализации, отличающихся третьим этапом. Реализации отличаются количеством параллельно выполняемых процессов (нитей) и вычислительной сложности каждого из этих процессов.

В случае первой параллельной реализации одной нитью выполняются вычисления для всех пар углов θ, φ одной точке трехмерного пространства. Альтернативой является выбор в качестве вычислений на одной нити всех значений ρ для всех точек, но только для одной пары углов θ, φ . Из минусов второй реализации следует отметить многократные обращения в глобальную память GPU для чтения координат трехмерной точки. Однако, для обеих реализаций сложно оценить возникающие коллизии, когда для выполнений транзакции разных нитей необходимо изменить содержание одной и той же ячейки памяти.

Параллельная реализация № 1	Параллельная реализация № 2
Начальные данные: Облако точек **Результирующие данные:** Значения аккумуляторного массива для всех возможных пар углов θ, φ одной точки	**Начальные данные:** Облако точек, **Результирующие данные:** Значения аккумуляторного массива для всех точек одной пары углов θ, φ
Вычисление индекса нити id : $id = blockIdx.x * blockDim.x + threadIdx.x$	Вычисление индекса нити id : $id = blockIdx.x * blockDim.x + threadIdx.x$
Чтение из глобальной памяти трёхмерной координаты	Вычисление θ, φ : $\theta = id / 360$ и $\varphi = id \% 360$
Цикл по всем углам θ от 0 до π с шагом $\pi / 180$	Цикл по всем точкам облака данных (x_0, y_0, z_0)
Цикл по всем углам φ от 0 до π с шагом $\pi / 360$	Чтение из глобальной памяти трёхмерной координаты
Вычисление ρ	Вычисление ρ :
Приведение к целому типу ρ	Приведение к целому типу ρ
Если $\rho < \Delta$	Если $\rho < \Delta$
Атомарная операция инкремента: $A(\theta, \phi, \rho)$	Атомарная операция инкремента: $A(\theta, \phi, \rho)$
Конец цикла по углу φ	Конец цикла по точкам облака данных
Конец цикла по углу θ	

Как можно заметить из псевдокодов параллельных реализаций, одной нитью выполняются циклы по разным параметрам, с разным количеством операций. Размер сетки также отличается.

Ускорение параллельных реализаций по сравнению с последовательной вычислялось по формуле:

$$s = \frac{t_{CPU}}{t_{HtoD} + t_{\text{kernel}} + t_{DtoH}}, \quad (3)$$

где

t_{CPU} – время работы последовательного алгоритма,

t_{HtoD} – время пересылки входных данных из оперативной памяти CPU в глобальную память GPU,

t_{kernel} – время работы ядра CUDA,

t_{DtoH} – время пересылки результатов работы ядра GPU (kernel) из глобальной памяти GPU в память CPU.

5. Результаты экспериментов

Для проверки эффективности CUDA-реализаций параллельного алгоритма были проведены следующие эксперименты. В качестве входных данных использовалось облако точек из 158877 точек. Эксперименты проводились на следующем оборудовании: CPU: Intel Core i7-6700K, 4 GHz, GPU: GeForce GTX 750 Ti. Результаты сравнительных исследований времени реализаций алгоритма приведены в таблице 1.

Последовательная реализаций алгоритма Хафа выполнилась за 13 секунд. Для параллельных реализаций время выполнения составило 1353 и 2139 секунд соответственно. Особенность параллельных реализаций заключается в возникновениях ситуаций, когда разные нити одновременно выполняют операцию инкремента над одной и той же переменной. Для атомарного доступа каждой нити к определенной области памяти использовалась специальная операция, гарантирующая атомарность *atomicAdd()*. Наименьшее время выполнения было установлено для

параллельной реализации №1, для которой параллелизм осуществлялся на уровне декомпозиции по данным облака точек.

Таблица 1. Время выполнения и ускорение

	Время выполнения (миллисекунды)	Ускорение
Последовательная реализация	13098	–
Параллельная реализация № 1	1353	9,7
Параллельная реализация № 2	2139	6,1

6. Заключение

Предложенный алгоритм был реализован в виде программы на языке C++ с использованием технологии CUDA. Проведены экспериментальные исследования по достижимым показателями точности и надёжности. В ходе экспериментального исследования технологии была показана её работоспособность и проведено сравнительное исследование эффективности различных параллельных программных реализаций предложенного алгоритма. Наибольшее ускорение получено для параллельной реализации № 1 в 9,7 раз.

Благодарности

Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект № 16-07-00729 а).

Авторы выражают благодарность д.т.н., профессору Фурсову В.А. за конструктивное обсуждение и ряд важных замечаний при подготовке настоящей работы.

Литература

[1] Pollefeys, M. Detailed real-time urban 3d reconstruction from video / M. Pollefeys, D. Nistér, J.M. Frahm, A. Akbarzadeh //International Journal of Computer Vision. – 2008. – Vol. 78(2-3) – P. 143-167.

[2] Baillard, C. 3-D reconstruction of urban scenes from aerial stereo imagery: a focusing strategy / C. Baillard, H. Maıtre //Computer Vision and Image Understanding. – 1999. – Vol. 76(3). – P. 244-258.

[3] Pollefeys, M. Self-calibration and metric reconstruction inspite of varying and unknown intrinsic camera parameters / Pollefeys M., Koch R., Van Gool L. //International Journal of Computer Vision. – 1999. – Vol. 32(1). – P. 7-25.

[4] Eisert, P. Automatic reconstruction of stationary 3-D objects from multiple uncalibrated camera views / Eisert P., Steinbach E., Girod B. // Circuits and Systems for Video Technology, IEEE Transactions on. – 2000. – Vol. 10(2). – P. 261-277.

[5] Reitberger, J. 3D segmentation of single trees exploiting full waveform LIDAR data / J. Reitberger, C. Schnörr, P. Krzystek, U. Stilla // ISPRS Journal of Photogrammetry and Remote Sensing. – 2009. – Vol. 64(6). – P. 561-574.

[6] Tarsha-Kurdi, F. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data / F. Tarsha-Kurdi, T. Landes, P. Grussenmeyer // Proceedings of the ISPRS Workshop on Laser Scanning. – 2007. – Vol. 36. – P. 407-412.

[7] Zhang, J. SVM-based classification of segmented airborne LiDAR point clouds in urban areas / Zhang J., Lin X., Ning X. // Remote Sensing. – 2013. – Vol. 5(8). – P. 3749-3775.

[8] Borrmann, D. The 3D Hough Transform for plane detection in point clouds: A review and a new accumulator design / D. Borrmann, J. Elseberg, K. Lingemann, A. Nüchter // 3D Research. – 2011. – Vol. 2(2). – P. 1-13.

[9] Goshin, Ye.V. Segmentation of stereo images with the use of the 3D Hough transform / Goshin Ye.V., Loshkareva G.E. // CEUR Workshop Proceedings. – Vol. 1638 – 2016. – P.340-347.

[10] Lucas, B. D. An iterative image registration technique with an application to stereo vision / B.D. Lucas, T. Kanade // IJCAI. – 1981. – Vol. 81. – P. 674-679.

[11] Hartley, R.I. Triangulation / Hartley R. I., Sturm P. // Computer vision and image understanding. – 1997. – Vol. 68(2). – P. 146-157.

[12] Якимов, П.Ю. Локализация контуров объектов на изображениях при вариациях масштаба с использованием преобразования Хафа / Фурсов В.А., Бибиков С.А., Якимов П.Ю. // журнал Компьютерная оптика. – 37(4). – 2013. – С. 496-502.

[13] Разлацкий, С.А. Исследование методов распознавания объектов в трехмерной сцене / Разлацкий С.А., Якимов П.Ю. // труды международной научно-технической конференции ПИТ 2015. – 2015. – С. 197-201.

[14] Van Den Braak, G. J. GPU-vote: A framework for accelerating voting algorithms on GPU / G. J. Van Den Braak, C. Nugteren, B. Mesman, H. Corporaal //European Conference on Parallel Processing. – Springer Berlin Heidelberg. – 2012. – P. 945-956.

[15] Руководство по программированию. NVIDIA Corporation: NVIDIA CUDA C Programming Guide - Version 8.0 (January 2017) [Электронный ресурс]. – Режим доступа: http://opencv.org http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf (01.01.2017).