УДК 004.89, 004.93

К.А. РУЧКИН, канд. физ.-мат. наук, доц., ГВУЗ "ДонНТУ", Донецк,
Е.А. ШЕВЧЕНКО, асп., ГВУЗ "ДонНТУ", Донецк

ПОВЫШЕНИЕ ЭФФЕКТИВНОСТИ ОБОБЩЕННОГО 3D ПРЕОБРАЗОВАНИЯ ХАФА

В данной работе продолжены исследования, связанные с повышением эффективности обобщённого преобразования Хафа для поиска нескольких сферических поверхностей по заданному трехмерному массиву точек в пространстве. Предложен способ распараллеливания работы алгоритма для ускорения его работы на многоядерных персональных компьютерах. Наибольшую эффективность данный поход показывает вместе с каскадным преобразованием. Ил.: 1. Табл.: 1. Библиогр.: 10 назв.

Ключевые слова: 3D преобразование Хафа, каскадное преобразование, распараллеливание работы алгоритма.

Постановка проблемы. Задача детектирования простых геометрических фигур на изображении является одной из широко известных задач автоматизированной обработки изображений. С этой задачей достаточно успешно справляется преобразование Хафа. Классический алгоритм Хафа был предназначен для нахождения прямой на изображении, но позже он был обобщен и на другие геометрические фигуры. Теперь его можно применять для обнаружения произвольных как параметризируемых объектов (например, эллипсов, окружностей, парабол), так и не параметризируемых объектов (с произвольной формой). По сути алгоритм Хафа является алгоритмом поиска и подбора параметров, значения которых в дискретном пространстве поиска будут соответствовать параметрам исходной модели. Однако чрезмерная дискретизация пространства или увеличение количества объектов поиска существенно увеличивают время работы алгоритма, делая его неэффективным. Эта проблема является одной из актуальных проблем и в настоящее время. Для ее решения предложены различные модификации по уменьшению и сокращению пространства поиска, основанные на вероятностных характеристиках пространства поиска.

В последнее время все чаще появляются различные обобщения преобразования Хафа на трехмерный случай. Однако при поиске трехмерных объектов увеличивается размерность аккумуляторного массива, что снова приводит к увеличению времени работы алгоритма. Повышению эффективности работы трехмерного преобразования Хафа за счет использования многопоточного подхода посвящена эта работа.

Анализ литературы. Исследованием преобразования Хафа занимались многие ученые. Отметим работы [1 – 6].

Преобразование Хафа в классическом виде, проблемы реализации в мультипроцессорных системах, а также особенности реализации на графических процессорах (GPU) описаны в [2]. Приведены иллюстрации работы метода, а также описан более обобщённый алгоритм распознавания произвольного образа.

В работе [3] обсуждается новый метод распознавания окружностей с помощью измененного преобразования Хафа. В основе метода лежит идея сокращения трехмерного накопительного массива (для распознавания окружности нужны 3 параметра) до двух двумерных, что позволяет несколько сократить объем вычислений и значительно экономить оперативную память. Также обсуждается процесс реализации распараллеливания данного алгоритма с помощью машины MIMD (сеть реконфигурируемых транспьютеров) и сравнение этой реализации по скорости выполнения со стандартным подходом.

В работе [4] обсуждается процесс нахождения произвольного объекта на изображении с помощью быстрого обобщенного преобразования Хафа, используя распараллеливание с помощью ресурсов GPU (применяя технологию CUDA). Рассмотрены основные проблемы распараллеливания, испытаны несколько вариантов реализации (максимальное распараллеливание или максимизирование времени владения общими ресурсами). Приведена таблица и сравнительные графики для каждого метода, предложены методы уплотнения, сортировки, репликации памяти, которые помогают оптимизировать разработанный алгоритм.

В статье [5] описано применение преобразования Хафа для поиска окружностей на изображении. Автором в базовый алгоритм были внесены некоторые изменения, которые позволили значительно ускорить обработку изображения и заполнение аккумуляторного массива, что, в свою очередь, в несколько раз ускорило алгоритм. Главными направлениями оптимизации стали распараллеливание некоторых этапов вычислений и уменьшение в несколько раз исходного изображения.

В [6] изложены сведения о параллельном программировании с использованием технологии OpenMP для современных параллельных высокопроизводительных вычислительных систем с общей памятью.

Цель статьи – повышение эффективности использования обобщенного преобразования Хафа за счет распараллеливания алгоритма и использования мультипоточного подхода.

Классическое преобразование Хафа использует преобразование координат точек искомого объекта в точки накопительного пространства. Преобразование Хафа для поиска любого объекта как на плоскости (изображении), так и в пространстве можно коротко описать следующими шагами.

1. Выбор размерности накопительного пространства и его создание.

2. Выбор количества промежуточных значений при переводе точек из декартовой плоскости в накопительное пространство.

3. Перевод каждой граничной точки изображения (пространства) из декартовой системы в накопительное пространство.

4. Поиск максимума в накопительном пространстве.

5. Восстановление координат и размеров искомого объекта по найденному максимуму.

Наиболее ресурсоемким шагом является третий. Если посчитать сложность этого шага для поиска прямой на изображении, то выйдет следующая зависимость:

$$S = K \cdot N \cdot R, \qquad (1)$$

где S – рассчитываемая сложность; K – количество граничных точек; N – количество промежуточных углов прямой к оси OX (шаг угла в алгоритме); R – количество промежуточных расстояний перпендикуляра до центра координат (шаг расстояния в алгоритме).

В более простом виде эту зависимость можно выразить как $S = K \cdot N^2$, если $N = R$. Для более сложных фигур эта зависимость будет ещё более ресурсоемкой. Например, при поиске окружности её можно записать в виде $S = K \cdot A \cdot B$, где A, B – размеры окрестности по осям X и Y, в пределе которой ищется радиус.

То есть увеличение количества граничных точек приводит к полиномиальному увеличению сложности работы алгоритма и увеличению времени его работы.

Методы оптимизации алгоритма. Ускорение работы (уменьшение времени работы) классического алгоритма можно выполнить несколькими способами [7 – 9].

1. Уменьшение количества точек исходной выборки.

2. Уменьшение количества промежуточных вычислений (например, увеличение шага угла и шага расстояния при поиске линии).

3. Распараллеливание некоторых веток алгоритма.

Первый вариант был успешно исследован и описан как вероятностное преобразование Хафа. Суть метода состоит в том, чтобы случайным образом выбрать определенный процент граничных точек из

исходной выборки и уже по этим точкам провести исследование. То есть пропорционально уменьшению количества точек будет расти скорость алгоритма. Экспериментальным путем N. Kiryati и др. [7] было выяснено, что для достаточно больших выборок эффективность этого метода была уже при 2% отобранных точек из общего их числа. С другой стороны, в методе есть некоторый недостаток, и состоит он в проблеме вычисления этого необходимого минимального процента [8]. То есть, чтобы подобрать процент правильно, мы должны знать характер данных выборки (зашумленность, количество точек и т.п.). Если же пользоваться универсальным правилом выбора процентного соотношения, то это не даст достаточного прироста быстродействия.

Второй вариант так же может успешно применяться. Но, во-первых, при значительном увеличении шага мы уже не сможем достаточно точно обнаружить координаты искомого объекта, в некоторых крайних случаях это становится практически невозможно и шаг приходится уменьшать. Во-вторых, не разработано универсального алгоритма, позволяющего определить оптимальный шаг с точки зрения эффективного нахождения объекта и скорости его выполнения.

Третий вариант является одним из наиболее интересных и перспективных. Он не обладает недостатками двух предыдущих (не теряется точность вычислений с уменьшением выборки и не нужно подбирать эффективный шаг) и его эффективность зависит только от аппаратной части оборудования, на котором он будет выполняться. Но у этого метода есть свои сложности на этапе реализации [9].

Распараллеливание обобщённого преобразования Хафа. Опишем реализацию распараллеливания метода для поиска сферы в трехмерном пространстве. Для поиска произвольных объектов эту методику можно так же успешно применять.

В случае поиска сферы с помощью преобразования Хафа, алгоритм можно представить в следующем виде (радиус сферы постоянный):

1. *Задать точность вычислений (количество промежуточных углов) при поиске точек сферы U1, U2.*

2. *Задать радиус искомой сферы Rs.*

3. *Задать количество точек M облака данных, в котором ищется объект.*

4. *Создать массив облака точек данных Data[M][3].*

5. *Заполнить массив Data[M][3].*

6. *Задать размерность аккумуляторного массива – Ax, Ay, Az.*

7. *Создать аккумуляторный массив центров сфер – Acum[Ax][Ay][Az].*

8. *Обнулить аккумуляторный массив Acum[Ax][Ay][Az].*

9. *Цикл по всем точкам облака данных d от 1 до M с шагом 1:*

Цикл по всем углам AngleAlpha от 0 до PI с шагом PI/ U1:

*Цикл по всем углам AngleBeta от 0 до 2*PI с шагом 2*PI/U2:*

```
LXc = Data[d][1] + Rs *sin(AngleAlpha)*cos (AngleBeta);
LYc = Data[d][2] + Rs*sin(AngleAlpha)*sin(AngleBeta);
LZc = Data[d][3] + Rs*cos(AngleAlpha);
Acum[LXc][LYc][LZc] = Acum[LXc][LYc][LZc] +1;
```

Конец цикла по углу AngleBeta;

Конец цикла по углу AngleAlpha;

Конец цикла по всем точкам облака данных.

15. *В аккумуляторном массиве Acum[Ax][Ay][Az] найти ячейку с наибольшим значением.*

16. *По индексам i, j, k найденной ячейки Acum[i][j][k] восстановить координаты центра сферы:*

Xs = i; Ys = j; Zs = k.

18. *Отобразить найденную сферу в пространстве.*

Как видно, наиболее интересным с точки зрения распараллеливания является девятый шаг. Наиболее сложными проблемами в процессе распараллеливания алгоритма являются такие.

1. Поиск операций, которые можно распараллелить.

2. Выбор ресурсов, которые должны использоваться по отдельности каждым независимым потоком.

3. Проблема использования общих ресурсов.

Так как процесс создания параллельных потоков достаточно ресурсоемкий, то его было решено использовать минимум раз. В связи с этим распараллеливание применимо к самому верхнему циклу девятого шага, то есть граничные точки разделены поровну между потоками. В связи с этим все локальные ресурсы (углы – *AngleAlpha*, *AngleBeta*, шаг углов, переменные *LXc, LYc, LZc*) были выделены отдельно каждому потоку. Алгоритм может быть распараллелен на любое количество потоков, но не большее чем количество точек. С практической точки зрения, наибольшая эффективность алгоритма будет в том случае, когда количество потоков программы будет соответствовать количеству процессорных ядер вычислительной машины.

Наиболее важным вопросом оказалось использование общих ресурсов. В данном случае это был накопительный трехмерный массив. В связи с этим были исследованы два возможных варианта его использования.

1. Создание для каждого потока своего накопительного массива с целью предотвращения проблем одновременного доступа к ячейке. После выполнения всех потоков соответствующие ячейки каждого из массивов складываются и заносятся в главный накопительный массив.

2. У всех потоков накопительный массив – общий ресурс. Используя ограничения на одновременный доступ можно использовать этот общий ресурс. Но в данном случае если одновременно двум потокам нужно будет записать значение в одну ячейку массива, то один из них будет ожидать, пока второй выполнит операцию.

У каждого из методов были выявлены свои достоинства и недостатки. У первого метода можно выделить следующие особенности.

1) Неэффективное использование оперативной памяти. Пропорционально увеличению потоков увеличивается её использование. Это так же ограничивает использование данного алгоритма на ПК с малым объемом оперативной памяти.

2) Требуется некоторое время на выделение памяти и её освобождение.

3) Требуется время на объединение результатов.

4) Нет проблем одновременного доступа к одной ячейке.

У второго метода выявлены следующие особенности.

1) Количество занимаемой оперативной памяти практически равно обычному линейному алгоритму (за исключением дополнительного выделения для некоторых независимых ресурсов).

2) После завершения работы потоков накопительный массив не требует постобработки.

3) Время на выделение и освобождение памяти не увеличивается по сравнению с линейным алгоритмом.

4) Возможны задержки внутри потоков из-за одновременного доступа к одной ячейке накопительного пространства.

Результаты исследования отображены в сравнительной табл. 1. Тестирование проводилось на компьютере с такими характеристиками: операционная система – Windows 7 SP1 64bit, процессор – AMD Phenom II 4 ядра по 3.4 GHz, 4GB RAM. Как видно, использование своего накопительного массива для каждого потока выявилось неоправданным. Даже с ростом объема выборки этот метод так и не смог догнать метод с использованием общего накопительного массива. Таким образом, согласно исследованию можно сделать вывод, что задержки времени на выделение дополнительных ресурсов (памяти) всегда больше, чем задержки, связанные с одновременным доступом к общему ресурсу. Также негативным фактором является использование первым методом огромного количества оперативной памяти. Так если

второму алгоритму при его работе выделяется не более 500 МБ оперативной памяти (накопительный массив 500·500·500, тип данный – long integer), то для первого случая необходимо в 4 раза больше памяти (при использовании 4 параллельных потоков).

Таблица 1

Сравнительный анализ эффективности работы алгоритмов при поиске сферы в облаке точек

Кол-во элементов (точек пространства)	Время выполнения (сек.)		
	1 поток	4 потока (отдельные накопительные массивы)	4 потока (общий накопительный массив)
500	4,74	5,57	1,59
1000	9,3	6,73	2,76
10000	94,4	29,7	26,28
15000	128,33	39,22	36,63
20000	171	51,6	46,67

На рис. отображены данные из табл. 1. Согласно им можно сделать вывод, что наблюдается нелинейная зависимость времени выполнения от размера входящей выборки. Так же наблюдается нелинейная зависимость производительности от количества потоков выполнения, но с одной оговоркой. Если количество потоков программы больше чем, количество возможных параллельных потоков процессора, то роста производительности не наблюдается, а может наблюдаться даже некоторое замедление (до нескольких десятков миллисекунд).

Выводы. Практические исследования в области применения многопоточных вычислений в преобразовании Хафа показали хорошие результаты, значительно увеличив скорость выполнения алгоритма. Особенно это актуально в связи с ростом в ПК количества вычислительных ядер. Для оптимизации базового алгоритма в работе рассмотрены различные модификации преобразования Хафа. Рассмотрена применимость для поиска различного типа объектов. Разработан метод распараллеливания некоторых ветвей алгоритма, позволяющего ускорить алгоритм на многоядерных процессорах. Также улучшен метод поиска нескольких сфер в одном накопительном пространстве.

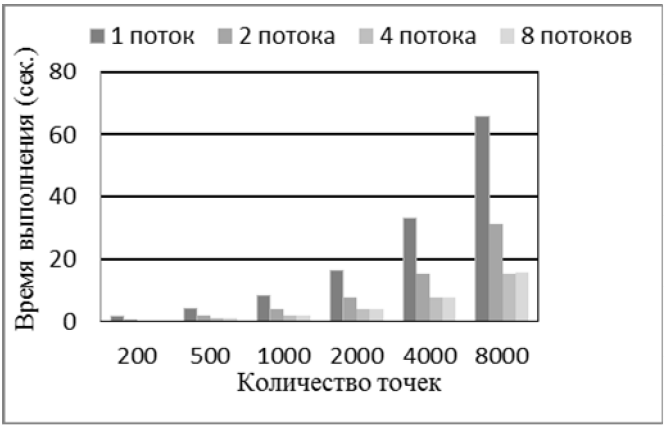

Рис. Зависимость времени выполнения программы от размера выборки и количества потоков

Разработано ПО, которое позволяет находить в облаке точек трехмерного пространства одну или несколько сфер за счет многопоточного режима. Среди недостатков данного ПО можно отметить необходимость ручного ввода радиуса искомой сферы. Среди достоинств можно отметить возможность загрузки произвольного файла данных и удобный, интуитивно понятный интерфейс.

Также положительной стороной исследуемого алгоритма распараллеливания является его применимость к разным вариантам обобщённого преобразования Хафа. Этот алгоритм может быть оптимизирован и для поиска линий, окружностей, и для произвольных объектов. Применение мультипоточных вычислений для вероятностного преобразования Хафа ещё больше ускорит работу алгоритма по сравнению с классическим преобразованием Хафа.

Список литературы: **1.** *Tuytelaars Tinne* The Cascaded Hough Transform / *Tinne Tuytelaars, Marc Proesmans, Luc Van Gool*. – Esat Mi. – 1998 // [Електронний ресурс]. – Режим доступа: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.2686. **2.** *Van den Braak* Fast Hough transform on GPUs: exploration of algorithm trade-offs / *G.J. Van den Braak, C. Nugteren, B. Mesman, H. Corporaal* // Springer, Advances Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science. – 2011. – Vol. 6915. – P. 611-622. **3.** *Chan R.* New parallel Hough transform for circles / *R. Chan, W. C. Siu* // IEE PROCEEDINGS-E. – 1991. – Vol. 138. – № 5. – P. 335-344. **4.** *Gomez-Luna Juan* Parallelization of the Generalized Hough Transform on GPU / *Juan Gomez-Luna, Jose Maria Gonzalez-Linares, Jose Ignacio Benavides, Emilio L. Zapata, Nicolas Guil*. – XXII Jornadas de Paralelismo, 2011. [Електронний ресурс]. – Режим доступа: http://jp2011.pcg.ull.es/sites/jp2011.pcg.ull.es/ files/Load.pdf **5**. *Денисюк В.С.* Применение и оптимизация преобразования Хафа для поиска объектов на изображении / *В.С. Денисюк*

// Межд. конгресс по информатике: информационные системы и технологии: Мат. межд. научного конгресса 31 окт. – 3 нояб. 2011 г.: в 2 ч. Ч. 2. – Минск: БГУ, 2011. – С. 162-165. **6.** *Антонов А.С.* Параллельное программирование с использованием технологии OpenMP: Учебное пособие. – М.: Изд-во МГУ, 2009. – 77 с. **7.** *Kiryat N.* A probabilistic Hough transform / *N. Kiryati, Y. Eldar, A.M. Bruckstein* // Pattern Recognition. – 1991. – Vol 24. – Issue 4. – P. 303-316. **8.** *Matas J.* Progressive Probabilistic Hough Transform / *J. Matas, C.Galambos, J. Kittler* // British Machine Vision Conference. – London, 1998. – Vol. 1. – P. 256-265. **9.** *Khoshelham K.* Extending Generalized Hough Transform to detect 3D objects in laser range data / *K. Khoshelham* // IAPRS. – 2007. – Vol. XXXVI. – Part 3. – P. 206-210. **10.** *Ручкин К.А.* Метод обнаружения окружностей в пространстве по трехмерному массиву данных / *К.А. Ручкин* // Интеллектуальные системы принятия решений и проблемы вычислительного интеллекта: Мат. межд. научной конференции // Херсонский НТУ. – Херсон, 2012. – С. 405-406.

Bibliography (transliterated): 1. *Tuytelaars Tinne* The Cascaded Hough Transform / *Tinne Tuytelaars, Marc Proesmans, Luc Van Gool* . – Esat Mi. – 1998 // [Elektronnij resurs]. – Rezhim dostupa: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.2686. **2.** *Van den Braak G.J.* Fast Hough transform on GPUs: exploration of algorithm trade-offs / *G.J. Van den Braak, C. Nugteren, B. Mesman, H. Corporaal* // Springer, Advances Concepts for Intelligent Vision Systems, Lecture Notes in Computer Science. – 2011. – Vol. 6915. – R. 611-622. **3.** *Chan R.* New parallel Hough transform for circles / *R. Chan, W. C. Siu* // IEE PROCEEDINGS-E. – 1991. – Vol. 138. – № 5. – P. 335-344. **4.** *Gomez-Luna Juan* Parallelization of the Generalized Hough Transform on GPU / *Juan Gomez-Luna, Jose Maria Gonzalez-Linares, Jose Ignacio Benavides, Emilio L. Zapata, Nicolas Guil* . – XXII Jornadas de Paralelismo, 2011. [Elektronnij resurs]. – Rezhim dostupa: http://jp2011.pcg.ull.es/sites/jp2011.pcg.ull.es/files/Load.pdf **5.** *Denisjuk V.S.* Primenenie i optimizacija preobrazovanija Hafa dlja poiska ob"ektov na izobrazhenii / *V.S. Denisjuk* // Mezhd. kongress po informatike: informacionnye sistemy i tehnologii: Mat. mezhd. nauchnogo kongressa 31 okt. – 3 nojab. 2011 g.: v 2 ch. Ch. 2. – Minsk: BGU, 2011. – C. 162-165. **6.** *Antonov A.S.* Parallel'noe programmirovanie s ispol'zovaniem tehnologii OpenMP: Uchebnoe posobie. – M.: Izd-vo MGU, 2009. – 77 s. **7.** *Kiryat N.* A probabilistic Hough transform / *N. Kiryati, Y. Eldar, A.M. Bruckstein* // Pattern Recognition. – 1991. – Vol 24. – Issue 4. – P. 303-316. **8.** *Matas J.* Progressive Probabilistic Hough Transform / *J. Matas, C.Galambos, J. Kittler* // British Machine Vision Conference. – London, 1998. – Vol. 1. – P. 256-265. **9.** *Khoshelham K.* Extending Generalized Hough Transform to detect 3D objects in laser range data / *K. Khoshelham* // IAPRS. – 2007. – Vol. XXXVI. – Part 3. – P. 206-210. **10.** *Ruchkin K.A.* Metod obnaruzhenija okruzhnostej v prostranstve po trehmernomu massivu dannyh / *K.A. Ruchkin* // Intellektual'nye sistemy prinjatija reshenij i problemy vychislitel'nogo intellekta: Mat. mezhd. nauchnoj konferencii // Hersonskij NTU. – Herson, 2012. – S. 405-406.

Поступила (*received*) *25.03.2014*

Статью представил д-р техн. наук, проф. БелГУ Корсунов Н.И.

Ruchkin Konstantin., Ph.D., Associate Professor
Donetsk National Technical University
Str. Artema, 58, Donetsk, Ukraine, 83001
tel./phone: (062) 342-91-21, e-mail: C_ruchkin@mail.ru

Shevchenko Evgeniy, assistant
Donetsk National Technical University
Str. Artema, 58, Donetsk, Ukraine, 83001
tel./phone: (062) 342-91-21, e-mail: sheva_aka_sea@mail.ru