Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Пенкрат Андрей Олегович

Выпускная квалификационная работа бакалавра

Построение карты глубины по изображениям стереопары

Направление 010400
Прикладная математика и информатика

Научный руководитель,
кандитат технических наук,
доцент
Гришкин В. М.

Санкт-Петербург
2017

Содержание

Введение

Карта глубины представляет собой матрицу расстояний от стереокамеры до объекта для каждого пикселя одного из изображений.

Компьютерное стереозрение позволяет получать карты глубины посредством фиксированной системы из двух оптических камер. Это дешевле, а в некоторых задачах и эффективнее, чем активные системы, такие как Microsoft Kinect и Intel RealSence.

Стереозрение применяется в таких областях, как распознавание жестов, обход препятствий и 3D-реконструкция. Оно является областью активных исследований на протяжении многих лет, однако эффективное со всех точек зрения решение пока ещё не найдено.

Постановка задачи

Задача компьютерного стереозрения заключается в нахождении смещения каждого пикселя в заданном диапазоне $[d_{min}, d_{max}]$ на ректифицированной стереопаре изображений с целью получения карты глубины.

Ректификация — это преобразование, приводящее два или более изображений к одной общей плоскости. При этом эпиполярные линии ($x_L e_L$ и $x_R e_R$ на рис. 1) приходят к виду $y = const$, что существенно упрощает задачу сопоставления точек [1].

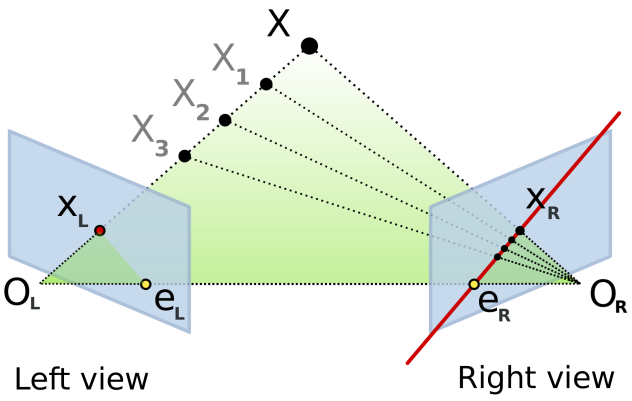

Рис. 1: Эпиполярная геометрия

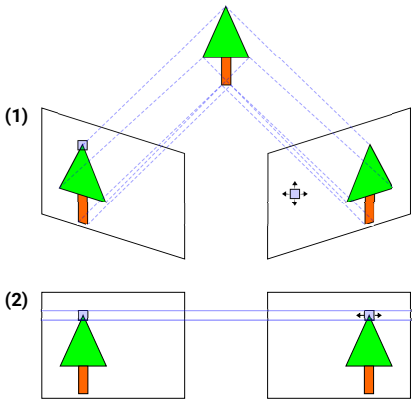

Рис. 2: Ректификация изображений

Найдя соответствие точек левого и правого изображений, получим диспаритеты (отклонения) каждого пикселя одного из них (обычно за основу берётся левое). Диспаритет связан с глубиной следующим отношением:

$$\frac{T-d}{Z-f} = \frac{T}{Z},$$

где T — расстояние между камерами (baseline), f — фокусное расстояние, d — диспаритет, а Z — глубина.

В данной работе ставится цель получения приемлемых по качеству карт диспаритетов в условиях, близких к реальному времени, то есть требуется обрабатывать 10 стереокадров в секунду.

Обзор литературы

В [1] даются основные понятия эпиполярной геометрии и её применение в компьютерном стереозрении.

В [2] проводится общий анализ алгоритмов стереозрения, в частности, формулируется обобщённая схема работы данных алгоритмов.

В [3] предлагается новый алгоритм стереозрения, использующий комбинацию градиентной метрики с облегчённой версией цензус-преобразования, а так же управляющее изображение, получаемое фильтрацией исходного стереокадра и влияющее на все этапы метода.

В [4] вводится алгоритм, основанный на комбинированной метрике AD-Census, крестовой агрегации, полуглобальной оптимизации и многошаговом улучшении диспаритетов.

В [5] представлен метод, включающий те же этапы, что [4], однако использующий нейронную сеть для сравнения фрагментов изображения и получения цен соответствия.

В [6] описан алгоритм, построенный на метрике взаимной информации и одномерной оптимизации энергетической функции по направлениям. Данный метод сильно повлиял на развитие направления и многие его элементы используются в более поздних работах.

Статья [7] описывает способ съёмки наборов данных для оценки алгоритмов стереозрения, используемый в широко известном бенчмарке Middlebury.

Глава 1. Основные подходы

В зависимости от конечной цели в стереозрении могут применяться различные подходы.

Для задачи распознавания жестов и некоторых других может быть достаточно «несплошной» карты диспаритетов, содержащей значения лишь для отдельных точек. Такую карту чаще всего получают методами, основанными на ключевых точках.

Более широкое применение и развитие имеют алгоритмы получения «сплошной» карты, покрывающей всё изображение. Такие методы, в свою очередь, традиционно делятся на локальные и глобальные.

Локальный алгоритм стереозрения — это алгоритм, в котором для сопоставления точек на правом и левом изображениях стереокамеры используются только определённые окрестности этих точек.

В глобальных же алгоритмах диспаритет каждого пикселя определяется изображениями целиком. Обычно это достигается построением и глобальной минимизацией так называемой «энергетической функции».

На практике современные алгоритмы зачастую комбинируют элементы обоих категорий. Глобальные методы строят энергетические функции на основе цен соответствия, полученных локальными методами, а локальные — производят ограниченную оптимизацию энергетической функции.

Основным предметом статьи будут локальные алгоритмы, как наиболее применимые в задачах реального времени.

1.1. Общая схема

Исследователи часто делят алгоритмы стереозрения на следующие четыре этапа [2]:

1. Подсчёт цен соответствия точек.
2. Агрегация цен.
3. Вычисление диспаритетов.
4. Уточнение диспаритетов.

1.2. Выбор алгоритма

Основным показателем при выборе алгоритма являлась позиция в оценках бенчмарка Middlebury версий 2 и 3.

В версии 2 лидирующие позиции занимают Image-Guided Stereo Matching [3] и AD-Census [4]. В третьей большинство алгоритмов с хорошими результатами являются глобальными, единственный локальный метод, входящий в топ 10 — MC-CNN [5] — основан на AD-Census, однако использует нейронную сеть для получения цен соответствия. Широко распространённый Semi-Global Matching [6] занимает существенно более низкие позиции в обоих версиях.

Выбор был остановлен на AD-Census, как на хорошо описанном параллельном алгоритме, проверенном в задачах реального времени.

Глава 2. AD-Census

В 2011 году группа учёных из Китая представила локальный алгоритм стереозрения, занявший первую позицию в бенчмарке Middlebury. Алгоритм основывается на метрике схожести AD-Census, крестовой агрегации и полуглобальной оптимизации цен.

2.1. Ценовая метрика

Метрика AD-Census составляется из абсолютной разности цветов и расстояния Хэмминга между цензус-преобразованиями с окном 9×7. Комбинируются данные величины суммой результатов робастной функции вида $1 - e^{-x/\lambda}$. Здесь $\lambda = \{\lambda_{AD}, \lambda_{Census}\}$ — коэфициент, задающий влияние каждой из величин.

Цена соответствия вычисляется для каждого пикселя базового (левого) изображения с диспаритетами в заданном диапазоне.

Цензус-преобразование — бинарный дескриптор, содержащий единицы для пикселей, яркость которых больше либо равна яркости центрального, и нули для меньшей яркости.

Расстояние Хэмминга — количество различающихся бит.

2.2. Агрегация

Далее цены сглаживаются усреднением по областям с близкими цветами.

Каждому пикселю ставится в соответствие крест, состоящий из пикселей, близких по цвету. Кресты строятся итеративно, пиксели добавляются к плечам, пока выполнены следующие критерерии:

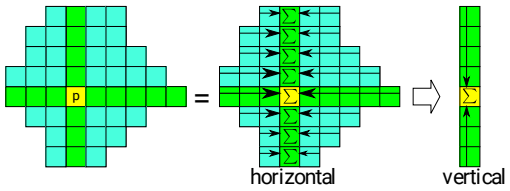

Рис. 3: Крестовая агрегация

1. $D_c(p_1, p) < \tau_1$ и $D_c(p_1, p_1 + (1, 0)) < \tau_1$;
2. $D_s(p_1, p) < L_1$;
3. $D_c(p_1, p) < \tau_2$, если $L_2 < D_s(p_1, p) < L_1$.

Здесь D_c — разница цветов, D_s — разница координат.

Далее производится агрегация (подсчёт среднего арифметического цен) сначала по горизонтали, а затем по вертикали, как показано на рис. 3.

Этот этап повторяется четыре раза, при этом для улучшения устойчивости чередуется порядок направлений (при втором и четвёртом проходах агрегация выполняется сперва по вертикали, затем по горизонтали).

2.3. Оптимизация по направлениям

Следующий этап производит полуглобальную оптимизацию по четырём направлениям методом, предложенным Хиршмюллером в рамках алгоритма SGM.

Цены $C_r(p, d)$ обновляются на основе агрегированных цен $C_1(p, d)$ следующим образом:

$$C_r(p,d) = C_1(p,d) + \\ + \min(C_r(p-r,d), C_r(p-r,d\pm 1) + P_1, \min_k C_r(p-r,k) + P_2) - \\ - \min_k C_r(p-r,k),$$

где $p-r$ — предыдущий пиксель текущего направления, P_1, P_2 — штрафы, снижающиеся в точках резкого изменения цвета.

Итоговая цена равна среднему арифметическому оптимизированных цен по четырём направлениям (два горизонтальных и два вертикальных).

2.4. Вычисление диспаритетов

Каждому пикселю присваивается диспаритет, удовлетворяющий минимуму цены:

$$d(p) = \arg \min_{d \in [d_{min}, d_{max}]} C(p,d).$$

2.5. Уточнение диспаритетов

Далее, полученные диспаритеты улучшаются в несколько шагов:

1. Все предыдущие этапы повторяются с правым изображением в качестве основного. Выделяются ошибочные точки, удовлетворяющие неравенству $d_L(x,y) \neq d_R(x - d_L(x,y), y)$.

2. Для ошибочных пикселей составляются гистограммы распределения диспаритетов верных (не ошибочных) пикселей в тех же областях, что

использовались при агрегации. В случае, если один из диспаритетов набрал более 40% голосов, а общее число голосов оказалось выше заданного порога, этот диспаритет назначается обрабатываемому пикселю и он более не считается ошибочным. Эта процедура повторяется пять раз, чтобы убрать большинство ошибок.

3. Оставшиеся ошибки исправляются в зависимости от своего типа. Если $\exists d : d = d_R(x - d, y)$, пикселю присваивается диспаритет самого близкого по цвету пикселя окрестности. В противном случае, пиксель считается принадлежащим фону и ему выбирается самое низкое значение диспаритета в области.

4. Для исправления неточностей на границах объектов, цена текущего диспаритета каждого пикселя сравнивается с ценами текущих диспаритетов своих соседей. Если цена диспаритета соседа оказалась ниже, пикселю присваивается диспаритет этого соседа.

5. Наконец, для достижения субпиксельной точности производится параболическая интерполяция

$$d^* = d - \frac{C(p, d+1) - C(p, d-1)}{2(C(p, d+1) + C(p, d-1) - 2C(p, d))}$$

и полученные диспаритеты сглаживаются медианным фильтром 3×3.

Глава 3. Реализация

В ходе работы была получена полная реализация алгоритма AD-Census на CUDA.

Каждый этап алгоритма выполняется параллельно. Агрегация ускорена при помощи техники префиксных сумм, в которой вычисляется матрица префиксов, элемент (x, y) которой содержит сумму цен всех цен для координат $(x_i \leqslant x, y_i = y)$.

```
__global__ void
prefixSumHorKernel(bool pad = true) {
    const int y = blockDim.x * blockIdx.x + threadIdx.x + WINDOW_HH;
    const int tidd = blockDim.y * blockIdx.y + threadIdx.y;
    const int disp = tidd + minDisparity;

    if(y >= ch - WINDOW_HH || disp > maxDisparity)
        return;

    float sum, cost;

    surf2DLayeredwrite(0, costsOut,
                       (WINDOW_WH+disp-1) * sizeof(float), y, tidd);

    surf2DLayeredread(&sum, costsIn,
                      (WINDOW_WH+disp) * sizeof(float), y, tidd);

    for(int x = WINDOW_WH + (pad ? disp : 0) + 1; x < cw - WINDOW_WH; x++) {
        surf2DLayeredread(&cost, costsIn,
                          x * sizeof(float), y, tidd);
        sum += cost;
        surf2DLayeredwrite(sum, costsOut,
                           x * sizeof(float), y, tidd);
    }
}
```

После этого можно быстро получить сумму цен в любом диапазоне, вычтя из префиксной суммы на верхней границе диапазона значение на единицу ниже нижней границы.

```
// Needs costsIn to be filled with horizontal prefix sums
```

```
__global__ void
costAggregationHorKernel(bool avg = true) {
    const int tidd = blockDim.z * blockIdx.z + threadIdx.z;
    const int disp = tidd + minDisparity;
    const int x = blockDim.x * blockIdx.x + threadIdx.x + WINDOW_WH + (avg ? disp :
    const int y = blockDim.y * blockIdx.y + threadIdx.y + WINDOW_HH;

    if(x >= cw - WINDOW_WH || y >= ch - WINDOW_HH || disp > maxDisparity)
        return;

    float cost;
    float prefix;
    uchar2 arms;

    // Read left and right arms into uchar2
    surf2Dread(&arms, surf2Dcrosses, x * sizeof(int), y);

    // If arms don't fit into image boundaries, cut them
    if(x - arms.x < (avg ? disp : 0) + WINDOW_WH)
        arms.x = x - (avg ? disp : 0) - WINDOW_WH;
    if(x + arms.y >= cw - WINDOW_WH)
        arms.y = cw - WINDOW_WH - x - 1;

    int len = arms.x + arms.y + 1;

    // Sum costs up using sum prefixes
    surf2DLayeredread(&prefix, costsIn,
                      (x + arms.y) * sizeof(float), y, tidd);
    cost = prefix;
    surf2DLayeredread(&prefix, costsIn,
                      (x - arms.x - 1) * sizeof(float), y, tidd);
    cost -= prefix;

    // Write average as a new cost
    if(avg)
        cost /= len;
    surf2DLayeredwrite(cost, costsOut,
                       x * sizeof(float), y, tidd);
}
```

Многие этапы также ускорены использованием SIMD-инструкций (Single Instruction, Multiple Data) и разделяемой памяти.

Глава 4. Модификация алгоритма

В процессе реализации и тестирования вышеописанного алгоритма выявились следующие недостатки в методе агрегации:

- очень близкие по цвету объекты, расположенные друг за другом, «сливаются»;
- на горизонтальных поверхностях в результате агрегации точек различных диспаритетов падает точность.

Для исправления данных недостатков предлагается изменение в методе составления агрегационных крестов, учитывающее исходные цены соответствия. Как и в исходном алгоритме, с каждой из четырёх сторон от пикселя в крест итеративно добавляются новые точки, пока не будет нарушено условие добавления либо превышен порог максимальной длины плеча. Однако условие добавления пикселя в крест заменяется следующим:

$$1 - e^{-||I(p)-I(p_1)||/\lambda_{AD}} + \\ + \min \left\{ \begin{array}{l} \left| \min\limits_d C(p,d) - C\left(p_1, \arg\min\limits_d C(p,d)\right) \right|, \\ \left| \min\limits_d C(p_1,d) - C\left(p, \arg\min\limits_d C(p_1,d)\right) \right| \end{array} \right\} \leqslant \tau_a.$$

Здесь p_1 — пиксель для добавления, τ_a — порог, который принимается равным 0,5 и уменьшается вдвое при длине плеча больше половины максимальной.

Таким образом изменение цены соответствия влияет на включение в крест агрегации точно так же, как и изменение цвета. Значение коэфициента τ_a подобрано эмпирически.

Выводы

Для оценки качества алгоритма использовался бенчмарк Middlebury с тренировочным набором данных, представляющим из себя набор ректифицированных стереоснимков различных сцен в трёх разрешениях и с точными результатами (полученными с помощью структурированного света) для сравнения [7]. Основными показателями является количество ошибочных пикселей и средняя ошибка. В качестве порога ошибки выбирались значения 1,0 и 0,5 пикселя для половинного и четвертичного разрешений соответственно. На полном разрешении оценка не производилась по причине высоких требований к видеопамяти.

Полученная реализация алгоритма на CUDA обрабатывает 5 кадров в секунду для половинного разрешения (1500×1000) и 10 кадров — для четвертичного (750×500) на GeForce GTX 1060. Это удовлетворяет требованиям задач реального времени, только на низком разрешении, однако предложенная модификация во многих случаях заметно улучшает качество без существенных потерь в производительности: не более 10 мс и 2 мс для половинного и четвертичного разрешений соответственно. Можно надеятся, что она позволит упростить более тяжёлые части алгоритма и получить приемлемое качество при выполнении в режиме 10 кадров в секунду даже на HD снимках и на более слабом оборудовании.

Сравнение качества модификации и исходного алгоритма приведено в таблицах 1 и 2.

Таблица 1: Четвертичное разрешение

Имя	Число ош.	Ч. О. оригинала	Средняя ош.	С. О. оригинала
Motorcycle	16,83	31,58	0,66	0,87
Piano	29,18	43,37	1,35	1,51
Pipes	19,31	24,51	1,43	1,87
PlaytableP	19,76	42,43	0,64	1,77
Recycle	22,27	37,29	0,62	0,72

Таблица 2: Половинное разрешение

Имя	Число ош.	Ч. О. оригинала	Средняя ош.	С. О. оригинала
Motorcycle	13,91	21,93	1,36	1,43
Piano	25,21	32,65	2,68	2,74
Pipes	16,02	17,84	2,52	3,19
PlaytableP	16,00	24,86	1,00	1,82
Recycle	21,95	26,40	2,29	1,51

Как видно, число ошибок существенно снизилось, особенно на четвертичном разрешении. Улучшение средней ошибки на части данных куда менее очевидно, что требует дополнительного исследования, и, возможно, исправлений в модификации.

Заключение

Предложенная модификация улучшает качество алгоритма без существенных потерь в скорости и итоговая реализация, запущенная на GeForce GTX 1060, соответствует требованиям задач реального времени на разрешении 750×500. В дальнейшем планируется упростить алгоритм для эффективной работы на большем разрешении и на более слабом оборудовании.

Список литературы

[1] Hartley R. I., Zisserman A. Multiple view geometry in computer vision. Second edition. Cambridge University Press, 2004.

[2] Scharstein D., Szeliski R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms // International Journal of Computer Vision. 2002. No 47. P. 7–42.

[3] Zhang Y., Gu Y., Huang K. et al. Accurate image-guided stereo matching with efficient matching cost and disparity refinement // IEEE Transactions on Circuits and Systems for Video Technology. 2016. Vol. 26. No 9. P. 1632–1645.

[4] Xing M., Xun S., Mingcai Z. et al. On building an accurate stereo matching system on graphics hardware // 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops). Barcelona: IEEE. 2011. P. 467–474.

[5] Jure Žbontar, Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches // JMLR 17(65):1-32, 2016.

[6] Hirschmüller H. Stereo processing by semiglobal matching and mutual information // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2008. Vol. 30. No 2. P. 328–341.

[7] Scharstein D., Hirschmüller H., Kitajima Y. et al. High-resolution stereo datasets with subpixel-accurate ground truth // In German Conference on Pattern Recognition (GCPR 2014). Münster: GCPR. 2014. P. 31–42.