МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Программирование графических процессоров»

Обработка изображений на GPU. Фильтры.

Выполнил: Е.С. Пищик
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

Условие

Цель работы. Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.

Вариант 2. Медианный фильтр.

Программное и аппаратное обеспечение

GPU via SSH:

- Compute capability: 2.1
- Name: GeForce GT 545
- Total Global Memory: 3150381056
- Shared memory per block: 49152
- Registers per block: 32768
- Warp size: 32
- Max threads per block: (1024, 1024, 64)
- Max block: (65535, 65535, 65535)
- Total constant memory: 65536
- Multiprocessors count: 3

CPU via SSH: Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
RAM via SSH: 16 Gb.
HDD via SSH: 471 Gb.
OS: Windows 10
OS via SSH: Ubuntu 16.04.6 LTS
IDE: Visual Studio Code
Compiler via SSH: nvcc

Метод решения

Проходим по каждому пикселю изображения, для каждого пикселя вычисляем окно при помощи радиуса, внутри окна сортируем массив значений при помощи гистограммы и находим медианный элемент при помощи префиксной суммы, устанавливаем значение каналов r, g, b для пикселя как медианные значения, а w - яркость оставляем прежней. Нахождение медианного элемента в окне производится по отдельности для каждого из каналов r, g, b.

Описание программы

Программа состоит из одного основного файла gpu.cu. Файл состоит из функций median, kernel, main. median - функция находит медианный элемент по гистограмме и количеству элементов в окне. main - основная функция программы, здесь происходит считывание входных данных, выделение памяти под изображение на host устройстве, выделение памяти под изображение на device устройстве (при помощи функций cudaMalloc), копирование данных с host на device (при помощи cudaMemcpy). При помощи функций cudaEventCreate, cudaEventRecord, cudaEventSynchronize происходит измерение времени работы алгоритма. После всех операций по выделению памяти, копированию векторов,

созданию переменных для подсчета времени запускается функция kernel на device устройстве. После выполнения алгоритма идет вывод в потом ошибок времени затраченного на алгоритм, очищение памяти на host устройстве и device устройстве (при помощи cudaFree). В самом начале мы создаем глобальную переменную tex, которая является нашей текстурой. Перед привязкой массива к текстуре мы выставляем различные параметры для текстуры, такие как нормализация координат, политика работы вне индексов изображения и т.д. При помощи cudaMallocArray и cudaMemcpyToArray мы копируем входное изображение в массив на device устройстве и при помощи cudaBindTextureToArray связываем массив и текстуру.

В функции ядра, мы просто вычисляем абсолютные номера потоков в двумерной сетке - переменные idx и idy. А также количество всех наших потоков в двумерной сетке - переменные offsetx и offsety. В самом ядре мы просто проходим по пикселям при помощи idx и idy, потом проходим окном для каждого пикселя и считаем гистограмму для r,g,b каналов, далее для каждого из каналов мы находим медианный элемент. Сохраняем новые значения в пиксель. При помощи tex2D(tex, x, y) мы получаем объект пикселя в виде uchar4 (x, y, z, w) из нашей текстуры tex.

```
__global__ void kernel(uchar4 *out, int w, int h, int r)
{
   int idx = blockDim.x * blockIdx.x + threadIdx.x;
   int idy = blockDim.y * blockIdx.y + threadIdx.y;
   int offsetx = blockDim.x * gridDim.x;
   int offsety = blockDim.y * gridDim.y;

   uchar4 p;

   int rhist[256], ghist[256], bhist[256];
   int red, green, blue, cnt, x_start, x_end, y_start, y_end;

   for(int y = idy; y < h; y += offsety) {
      for(int x = idx; x < w; x += offsetx) {
         cnt = 0;

         for(int i = 0; i < 256; ++i) {
            rhist[i] = 0;
            ghist[i] = 0;
            bhist[i] = 0;
         }

         x_start = (x - r >= 0) ? x - r : 0;
         x_end = (x + r < w) ? x + r : w - 1;
         y_start = (y - r >= 0) ? y - r : 0;
         y_end = (y + r < h) ? y + r : h - 1;
```

```
            for(int m = y_start; m <= y_end; ++m) {
               for(int k = x_start; k <= x_end; ++k) {
                  p = tex2D(tex, k, m);
                  rhist[p.x] += 1;
                  ghist[p.y] += 1;
                  bhist[p.z] += 1;
                  cnt += 1;
               }
            }

            red = median(rhist, cnt);
            green = median(ghist, cnt);
            blue = median(bhist, cnt);

            out[x + y * w] = tex2D(tex, x, y);
            out[x + y * w].x = red;
            out[x + y * w].y = green;
            out[x + y * w].z = blue;
         }
      }
   }
```

Результаты

CPU

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	2.997	3.864	10.471	15.208	48.194	156.389

GPU kernel<<<dim3(32, 1), dim3(32, 1)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	2.175	3.564	7.433	13.061	31.998	174.813

GPU kernel<<<dim3(1024, 1), dim3(1024, 1)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	3.086	5.610	9.132	17.408	32.671	237.134

GPU kernel<<<dim3(8, 8), dim3(8, 8)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	1.939	3.467	6.043	11.053	19.837	129.531

GPU kernel<<<dim3(16, 4), dim3(16, 4)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	1.902	3.413	6.232	11.039	20.882	144.892

GPU kernel<<<dim3(16, 8), dim3(16, 8)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	2.780	4.963	8.902	16.912	28.561	205.146

GPU kernel<<<dim3(32, 32) dim3(32, 32)>>>

h,w,r	936, 1664, 5	936, 1664, 7	1600, 2560, 7	1600, 2560, 10	4000, 6000, 5	4000, 6000, 15
Время, s	2.929	5.274	9.833	18.783	31.751	229.954

h,w,r = 4000,6000,1

Устройство	Время, s
CPU	33.605
GPU kernel<<<dim3(32, 1), dim3(32, 1)>>>	13.254
GPU kernel<<<dim3(1024, 1), dim3(1024, 1)>>>	6.048
GPU kernel<<<dim3(8, 8), dim3(8, 8)>>>	5.967
GPU kernel<<<dim3(16, 4), dim3(16, 4)>>>	6.102
GPU kernel<<<dim3(16, 8), dim3(16, 8)>>>	5.212
GPU kernel<<<dim3(32, 32) dim3(32, 32)>>>	5.338

original h,w,r = 936,1664,5

filtered h,w,r = 936,1664,5

original h,w,r = 1600,2560,7

filtered h,w,r = 1600,2560,7

original h,w,r = 4000,6000,5

filtered h,w,r = 4000,6000,5

Выводы

В данной лабораторной работе я изучил различные виды памяти на GPU, в частности научился работать с текстурной памятью. Использовал двумерную сетку потоков, изучил медианный фильтр и применил его к картинкам разного разрешения с разными радиусами. Скорость работы алгоритма зависит очень сильно от размера окна, где мы ищем медианный элемент, чем меньше значение радиуса, тем быстрее отрабатывает GPU по сравнению с CPU. Думаю что проблема замедления GPU, относительно CPU при бОльших значениях радиуса связано с тем, что мы обработку пикселей делаем параллельно, но вот обработка окна в два цикла и нахождение медианного элемента для каждого окна происходит стандартным образом.