МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2
по курсу «Программирование графических процессоров»

Обработка изображений на GPU. Фильтры.

Выполнил: М.А. Жерлыгин
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

Цель работы: Научиться использовать GPU для обработки изображений.
Использование текстурной памяти.

Вариант №3: Выделение контуров. Метод Превитта(Прюитта).

Программное и аппаратное обеспечение

GPU:

1. Compute capability: 7.5;
2. Графическая память: 4294967296;
3. Разделяемая память: 49152;
4. Константная память: 65536;
5. Количество регистров на блок: 65536;
6. Максимальное количество блоков: (2147483647, 65535, 65535);
7. Максимальное количество нитей: (1024, 1024, 64);
8. Количество мультипроцессоров: 6.

Сведения о системе:

1. Процессор: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
2. Память: 16,0 ГБ;
3. HDD: 237 ГБ.

Программное обеспечение:

1. OS: Windows 10;
2. IDE: Visual Studio 2019;
3. Компилятор: nvcc.

Метод решения

Метод Превитта основан на обработке всех пикселей изображения двумя ядрами 3x3, которые сворачивают изображение, вычисляя приблизительные производные:
одну по горизонтали, другую по вертикали.

$$G_X = \begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix}$$

$$G_y = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$$

Чтобы получить значение градиента в точке достаточно взять среднеквадратичное значение градиентов по x и y.

$$G = \sqrt{G^2_X + G^2_y}$$

Описание программы

По условиям лабораторной используется текстурная ссылка для хранения пикселей. Единственное ядро использует двумерную сетку потоков и обрабатывает отдельные пиксели.
Значение градиента записывается в результирующий массив типа ***uchar4***.

```
__global__ void kernel(uchar4* out, uint32_t w, uint32_t h) {
  int idx = blockDim.x * blockIdx.x + threadIdx.x;
  int idy = blockDim.y * blockIdx.y + threadIdx.y;
  int offset_x = blockDim.x * gridDim.x;
  int offset_y = blockDim.y * gridDim.y;
  for (int x = idx; x < w; x += offset_x) {
      for (int y = idy; y < h; y += offset_y) {
          uchar4 z[9];
          z[0] = tex2D(tex, x - 1, y - 1);
          z[1] = tex2D(tex, x, y - 1);
          z[2] = tex2D(tex, x + 1, y - 1);
          z[3] = tex2D(tex, x - 1, y);
          z[4] = tex2D(tex, x, y);
          z[5] = tex2D(tex, x + 1, y);
          z[6] = tex2D(tex, x - 1, y + 1);
          z[7] = tex2D(tex, x, y + 1);
          z[8] = tex2D(tex, x + 1, y + 1);
          float result = op_prewit(z);
          if (result > 255) {
              result = 255;
          }
          unsigned char byte = result;
              out[x + y * w] = make_uchar4(byte, byte, byte,
z[4].w);
      }
   }
}
```

Результаты

Пример изображения:

Результат работы программы:

blocks	threads	time
16x16	16x16	5.450240
32x32	16x16	5.528064
8x8	8x8	5.688256
16x16	32x32	5.862272

64x64	16x16	5.932992
32x32	32x32	6.667456
4x4	4x4	21.764448
1024x1024	16x16	031.003839
1024x1024	32x32	204.964478
1x1	1x1	3699.413574

Выводы

Я ознакомился с обработкой изображений на GPU. На основе полученных измерений времени выполнения работы программы, можно сделать вывод, что при грамотном подборе конфигурации ядра, можно достигнуть улучшения скорости работы программы.

Конкретно этот алгоритм можно применить для подготовки данных для последующего обучения нейросети.