Программное обеспечение высокопроизводительных вычислительных систем

Лабораторная работа № 5

Применение графических процессоров для задач обработки изображений

Автор: Орлов Д.А.

Задачи обработки изображений – один из классов задач, для ускорения решения которых применяются графические процессоры (*GPU*). В данной лабораторной работе студентам предлагается изучить алгоритмы обработки полутоновых двумерных изображений и реализовать их в программах для центральных процессоров (*CPU*) и графических процессорах (*GPU*) архитектуры *NVIDIA CUDA*.

Программа должна загружать входное изображение из файла и записывать результат в другой файл. Таким образом, результаты работы программы можно будет просмотреть с помощью графического редактора.

Общие сведения о форматах файлов

В данной лабораторной работе входными данными должны быть полутоновые чёрно-белые изображения. Значение яркости каждого пикселя представлено беззнаковым целым восьмибитным числом (следовательно, диапазон изменения яркости от 0 до 255).

Обозначим через $I(x, y)$ яркость пикселя с координатами (x, y). Координаты изменяются соответственно от 0 до *W*-1 и от 0 до *H*-1, где *W* и *H* – ширина и высота изображения.

Изображение хранится в памяти ЭВМ в виде двумерного массива. Поэтому, если `data` – указатель на начало изображения, то `data[x+y*w]` содержит значение пикселя с координатами (*x*, *y*).

Для обмена данными можно использовать формат Targa. Функции чтения и записи в файл такого формата приведены в конце данного описания.

Задание на лабораторную работу

1) Реализовать алгоритм обработки изображения согласно варианту. Разработать программы для *CPU* (*host*) и *GPU* (*device*). Программы должны работать с изображениями произвольных разрешений до *FullHD* включительно (1920x1080

пикселей). Программа для *GPU* должна принимать на вход в качестве аргумента количество нитей, которые следует запустить на *GPU*.

2) Измерить времена выполнения алгоритмов на *GPU* и на *CPU* (учитывается только время выполнения алгоритма, времена загрузки данных из файла и записи данных в файл не учитываются). Для программы, реализующей вычисления на *GPU*, необходимо измерить не только время выполнения ядра, но и время выполнения алгоритма с учётом обмена данными между *device* и *host*.
3) Добиться ускорения обработки на *GPU*, с учётом обмена данными между *device* и *host*.

Указания к выполнению задания

1) Некоторые алгоритмы для вычисления значения яркости пикселя результирующего изображения требуют также значения яркости соседних пикселей исходного изображения. Чтобы получить результат для пикселей, находящихся на границах исходного изображения, надо доопредилить значения яркости для координат выходящих за границы изображения. Будем считать, что яркость такой точки равна яркости ближайшей точки изображения. Например, $I(-1,1) = I(0,1)$; $I(W,1) = I(W-1,1)$.
2) Поскольку яркость пикселя задаётся беззнаковым целым восьмибитным числом (тип данных unsigned char в C/C++), промежуточные вычисления следует проводить, используя типы float или int (в зависимости от задачи).
3) При записи вычисленных значений в результирующее изображение следует помнить, о том, что яркость пикселей результирующего изображения имеет тип unsigned char, а значит, может принимать значения только от 0 до 255. Поэтому перед записью результата отрицательные значения надо заменить на 0, а значения, большие 255 – на 255.
4) Для того, чтобы выполнить пункт 3 задания, необходимо запустить программу для *GPU* с разным конфигурациями выполнения (количеством блоков и нитей), подобрав ту, при которой достигается наибольшее ускорение. Кроме того, можно попробовать запустить программу для входных файлов разных размеров.

Варианты заданий:

1. Свёртка

Свёртка изображения с заданным ядром задаётся формулой:

$$R(x,y)=\sum_{j=-n}^{n}\sum_{i=-n}^{n}I(x+i,y+j)\cdot k(i,j),$$

где:

x, y – координаты пикселя

$R(x,y)$ – значение яркости пикселя результирующего изображения

$I(x,y)$ – значение яркости пикселя исходного изображения

k – ядро свёртки, как правило задаётся квадратной матрицей K, размерностью $N=2n+1$, причём, $k(i,j)=K(i+n,j+n)$.

Например, чтобы заменить значение яркости пикселя на средние значения яркости соседних пикселей, надо применить свёртку со следующим ядром: $\begin{pmatrix}0{,}125 & 0{,}125 & 0{,}125\\ 0{,}125 & 0 & 0{,}125\\ 0{,}125 & 0{,}125 & 0{,}125\end{pmatrix}$

Задание: запрограммировать алгоритм свёртки исходного изображения со следующим ядром: $\begin{pmatrix}-1 & -1 & -1\\ -1 & 9 & -1\\ -1 & -1 & -1\end{pmatrix}$. Это один из наиболее простых алгоритмов повышения чёткости изображения.

2. Свёртка с разделяемым ядром

Определение свёртки см. в описании варианта 1. Если матрицу K можно представить в следующем виде:

$$K(i, j) = F_1(i) \cdot F_2(j),$$

то говорят, что ядро *K* разделимо. Подставим эту формулу в определение свёртки:

$$R(x, y) = \sum_{j=-n}^{n} \sum_{i=-n}^{n} I(x+i, y+j) \cdot k(i, j) = \sum_{j=-n}^{n} \sum_{i=-n}^{n} I(x+i, y+j) \cdot f_1(i) \cdot f_2(j) =$$

$$= \sum_{j=-n}^{n} f_2(j) \sum_{i=-n}^{n} I(x+i, y+j) \cdot f_1(i)$$

здесь $f_1(i) = F_1(i+n)$, $f_2(j) = F_2(j+n)$

Это означает, что вычисление свёртки можно разбить на два этапа: свёртка по строкам и свёртка по столбцам. На первом этапе вычисляются промежуточные значения

$$M(x, y) = \sum_{i=-n}^{n} I(x+i, y) \cdot f_1(i),$$

на втором – вычисляется свёртка:

$$R(x, y) = \sum_{j=-n}^{n} M(x, y+j) \cdot f_2(j)$$

Такое разбиение позволяет ускорить процесс вычисления свёртки, так как этом случае потребуется всего 2*N* умножений на пиксель. В случае неразделимого ядра количество умножений составило бы N^2 на пиксель.

Задание: запрограммировать алгоритм Гауссова размытия изображения. Этот фильтр используется для подавления шума.

Этот фильтр реализуется свёрткой с разделяемым ядром следующего вида: $k(i, j) = g(i) \cdot g(j)$, где: $g(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$ – функция Гаусса, σ – радиус размытия. Для практических целей $g(x)$ считают равной нулю при $|x| > 6\sigma$. Поэтому, если для реализации алгоритма взять $\sigma = 3$, то размер ядра составит 19 на 19 (т.к. $3\sigma + 3\sigma + 1 = 19$).

3. Нерезкое маскирование (unsharp masking)

Нерезкое маскирование – это способ визуально сделать изображение более чётким. Основная идея состоит в вычитании из исходного изображения нерезкой составляющей (то есть, размытого изображения, умноженного на некоторый коэффициент). Яркость пикселя результирующего изображения вычисляется по следующей формуле:

$$I_U = I_O + a(I_O - I_B),$$

где:

I_U – яркость пикселя результирующего изображения (от unsharp),

I_O – яркость пикселя исходного изображения (от original),

I_B – яркость пикселя размытого изображения (от blurred),

a – коэффициент «силы» размытой составляющей (от amount), как правило $0 \le a \le 1$.

Таким образом, в результирующем изображении нерезкая составляющая будет замаскирована.

Задание: Реализовать алгоритм нерезкого маскирования. В качестве алгоритма размытия изображения использовать усреднение пикселей в квадратной окрестности размером 5 на 5 пикселей. Значение коэффициента a должно задаваться из командной строки.

4. Дискретное косинусное преобразование

Пусть F – матрица исходных пикселей изображения, размером N на N, C – результат дискретного косинусного преобразования, матрица частот, размером N на N. Тогда:

$$C = D \cdot F \cdot D^T,$$

где D – матрица коэффициентов дискретного косинусного преобразования, размером N на N. Коэффициенты матрицы D определяются по следующей формуле:

$$D_{ij} = k(j)\cos\left(j(i+0{,}5)\frac{\pi}{N} \right), \text{ где } k(j) = \begin{cases} \frac{1}{\sqrt{N}}, j = 0 \\ \frac{\sqrt{2}}{\sqrt{N}}, j \neq 0 \end{cases}$$

Обратное преобразование осуществляется по следующей формуле:

$$F = D^T \cdot C \cdot D$$

Дискретное косинусное преобразование применяется, например, в алгоритме сжатия *JPEG*. Коэффициенты, полученные после дискретного косинусного преобразования, подвергаются квантованию, то есть, делятся на коэффициенты матрицы квантования и округляются до ближайшего целого числа. За счёт этого происходит обнуление коэффициентов, отвечающих за высокочастотные составляющие сигнала, что позволяет достичь значительного сжатия данных при хранении таких коэффициентов. При декодировании сохранённые коэффициенты умножаются на коэффициенты матрицы квантования, а затем проводится обратное дискретное косинусное преобразование.

Если значение яркости каждого пикселя представлено беззнаковым целым числом, то перед выполнением преобразования из значения яркости вычитают половину максимального значения для данного диапазона (128 для восьмибитного значения яркости). Соответственно, после выполнения обратного преобразования, к полученному результату следует прибавить 128.

Задание: провести моделирование кодирования и декодирования по алгоритму JPEG. Для каждого блока изображения 8x8 пикселей:

- провести дискретное косинусное преобразование;
- провести квантование полученных коэффициентов согласно матрице:

$$\begin{pmatrix} 16, 11, 10, 16, 24, 40, 51, 61 \\ 12, 12, 14, 19, 26, 58, 60, 55 \\ 14, 13, 16, 24, 40, 57, 69, 56 \\ 14, 17, 22, 29, 51, 87, 80, 62 \\ 18, 22, 37, 56, 68, 109, 103, 77 \\ 24, 35, 55, 64, 81, 104, 113, 92 \\ 49, 64, 78, 87, 103, 121, 120, 101 \\ 72, 92, 95, 98, 112, 100, 103, 99 \end{pmatrix}$$

- провести обратное дискретное косинусное преобразование.

5. Выделение границ: оператор Робертса

Выделение границ на изображении применяется, например, для решения задач машинного зрения. Граница объекта на изображении характеризуется резким изменением значения яркости пикселя. Поэтому методы выделения границ в изображении основываются на вычислении градиента изображения.

Одним из самых простых методов выделения границ на изображении является оператор Робертса. Яркость пикселя результирующего изображения вычисляется по формуле:

$$R(x,y) = \sqrt{\left(I(x,y) - I(x+1,y+1)\right)^2 + \left(I(x+1,y) - I(x,y+1)\right)^2}$$

где:

x, y – координаты пикселя

$R(x,y)$ – значение яркости пикселя результирующего изображения

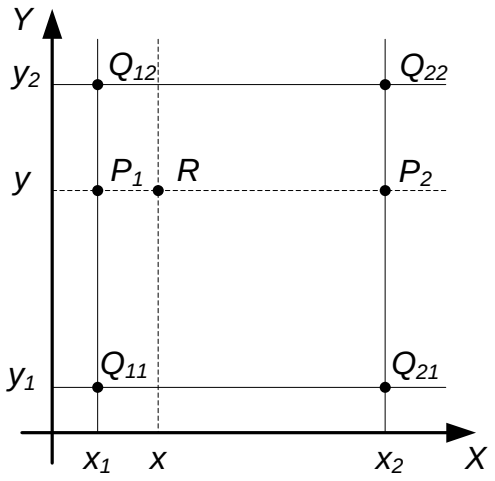
$I(x,y)$ – значение яркости пикселя исходного изображения

В результирующем изображении пиксели, находящиеся на границах областей с плавным изменением яркости, будут белыми. Остальные пиксели будут чёрными.

Задание: Реализовать алгоритм выделения границ, который использует оператор Робертса.

6. Увеличение изображения: билинейная интерполяция

Билинейная интерполяция – это расширение алгоритма линейной интерполяции на двумерный случай, когда интерполяция производится по одному измерению, а затем по другому.



Для того, чтобы узнать приближённое значение $F(R) = F(x, y)$, надо сначала вычислить приближённые значения $F(P_1)$ и $F(P_2)$. Они вычисляются методом линейной интерполяции:

$$F(P_1) = F(x_1, y) \approx \frac{y_2 - y}{y_2 - y_1} F(Q_{11}) + \frac{y - y_1}{y_2 - y_1} F(Q_{12})$$

$$F(P_2) = F(x_2, y) \approx \frac{y_2 - y}{y_2 - y_1} F(Q_{21}) + \frac{y - y_1}{y_2 - y_1} F(Q_{22})$$

Затем $F(R)$ вычисляется линейной интерполяцией значений $F(P_1)$ и $F(P_2)$:

$$F(R) = F(x, y) \approx \frac{x_2 - x}{x_2 - x_1} F(P_1) + \frac{x - x_1}{x_2 - x_1} F(P_2)$$

В простейшем случае увеличение изображения в *N* раз алгоритмом билинейной интерполяции проводится следующим образом: для каждой пары соседних пикселей в столбце исходного изображения вычисляются *N*-1 пикселей увеличенного изображения, затем, аналогичным способом вычисляются значения пикселей в строках. Таким образом, для исходного изображения размером *W* пикселей в ширину и *H* пикселей в высоту, интерполированное изображение будет иметь размер $(W-1)N+1$ пикселей в ширину и $(H-1)N+1$ пикселей в высоту.

Задание: реализовать алгоритм увеличения изображения в целое количество раз методом билинейной интерполяции.

7. Масштабирование Ланцоша (Lanczos)

Этот алгоритм масштабирования основан на фильтре Ланцоша, который применяется для передискретизации сигналов.

Пусть одномерный сигнал $f(x)$, известен в точках с целыми координатами. Тогда промежуточное значение $f(x+t), 0 \le t < 1$ можно получить из значений функции в нескольких соседних точках:

$$f(x+t) \approx \sum_{i=-n+1}^{n} f(x+i)h(t-i),$$

где $h(t)$ – функция Ланцоша.

$$h(t) = \begin{cases} 1, t = 0 \\ \sin c(t)\sin c\left(\frac{t}{n}\right), -n < t < n, \\ 0, \mathrm{t} \ge \mathrm{n} \cup \mathrm{t} \le \text{-n} \end{cases}$$

где $\sin c(t) = \frac{\sin(\pi t)}{\pi t}$.

Такой алгоритм интерполяции называют Lanczos-n. Как правило *n* выбирают равным 3. Интерполяция Lanczos3 используется в современных графических и видеоредакторах.

Не трудно показать, что при $n \to \infty$ выражение для $f(x+t)$ превращается в ряд Котельникова.

На двумерный случай фильтр обобщается следующим образом: сначала передискретизация происходит по одному измерению, затем – по другому.

Задание: реализовать алгоритм увеличения изображения в *N*=4 раза методом интерполяции Lanczos3.

8. Автоматическая регулировка яркости и контрастности

Пусть m и M – соответственно минимальное и максимальное значения яркости пикселя для данного изображения. Автоматическая регулировка позволит увеличить контраст изображения. Тогда, если для записи значения яркости отводится один байт, то диапазон изменения значений яркости пикселей будет $0 \le I \le 255$. Тогда яркость пикселей результирующего изображения будет вычислена по формуле:

$$I = \frac{(I_0 - m)255}{M},$$

где I_0 – яркость пикселя исходного изображения.

Задание: реализовать алгоритм автоматической регулировки контрастности, при этом в программе для графического процессора поиск максимального и минимального значений яркости пикселя также должен выполняться на графическом процессоре.

9. Масштабирование методом «до ближайшей соседней точки»

Пусть исходное изображение имеет размеры W_0 пикселей в ширину и H_0 пикселей в высоту. Пусть новое изображение имеет размеры W_1 пикселей в ширину и H_1 пикселей в высоту. Тогда пиксель нового изображения с координатами (i, j) будет иметь то же значение яркости, что пиксель исходного изображения с координатами $\left(\left[\frac{iH_0}{H_1}\right],\left[\frac{jW_0}{W_1}\right]\right)$.

Задание: реализовать алгоритм масштабирования изображения методом «до ближайшей соседней точки». На вход алгоритма помимо исходного изображения должны поступать размеры нового изображения. Размеры нового изображения могут быть произвольными: как больше, так и меньше размеров исходного изображения.

Функции записи и чтения данных из файлов формата Targa

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

//Сохраняет изображение в файл .tga
// path - путь к файлу
// data - указатель на данные изображения
// w - ширина изображения
// h - высота изображения
// bpp - количество бит на пиксель
// возвращает true, если удалось записать данные в файл, иначе - false
bool Array2Targa(const char *path, const unsigned char *data, unsigned w, unsigned h,
unsigned bpp){
    unsigned char TargaMagic[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
    if(bpp==8)
        TargaMagic[2]=3;
    else
        TargaMagic[2]=2;
    FILE *File = fopen(path, "wb");
    if(File==NULL){
        return false;
    }
    if(fwrite(TargaMagic,1,sizeof(TargaMagic),File)!= sizeof(TargaMagic)){
        fclose(File);
        return false;
    }
    unsigned char Header[6]={0};
    Header[0]=w&0xFF; Header[1]=(w>>8)&0xFF;
    Header[2]=h&0xFF; Header[3]=(h>>8)&0xFF;
    Header[4]=bpp;
    unsigned int ImageSize=w*h*(bpp)/8;
    if(fwrite(Header,1,sizeof(Header),File)!=sizeof(Header)){
        fclose(File);
        return false;
    }
    if(fwrite(data,1,ImageSize,File)!=ImageSize){
        fclose(File);
        return false;
    }
    fclose(File);
    return true;
}

//Загружает изображение из несжатого файла .tga
// path - путь к файлу
// pdata - указатель на переменную, в которую будет записан указатель на считанные
данные
//         обращаю внимание, что память для хранения данных надо освобождать вручную
// pw - указатель на переменную, в которую будет записана ширина изображения
// ph - указатель на переменную, в которую будет записана высота изображения
// pbpp - указатель на переменную, в которую будет записано количество бит на пиксель
// возвращает true, если удалось считать данные из файла, иначе - false
bool Targa2Array(const char *path, unsigned char **pdata, unsigned *pw, unsigned *ph,
unsigned *pbpp){
    const unsigned char TargaMagic[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
    unsigned char FileMagic[12];
    unsigned char Header[6];
    //char *data;
    FILE *File = fopen(path, "rb");
    if(File==NULL){
        return false;
    }
    if(fread(FileMagic,1,sizeof(FileMagic),File)!= sizeof(FileMagic)){
        fclose(File);
        return false;
```

```
    }
    unsigned char ImageType=FileMagic[2];
    FileMagic[2]=0;
    if(memcmp(TargaMagic,FileMagic,sizeof(TargaMagic))!= 0||
       fread(Header,1,sizeof(Header),File)!= sizeof(Header)){
        fclose(File);
        return false;
    }
    // Определяем размеры изображения
    *pw=Header[1]*256+Header[0];
    *ph=Header[3]*256+Header[2];
    // Определяем глубину цвета используемую в изображении
    *pbpp=Header[4];
    unsigned int Bpp=*pbpp/8;
    //Поддерживаются только изображения с 1, 3 или 4 байта на пиксель
    if(*pw<=0 || *ph<=0 || (ImageType==2&&Bpp!=3&&Bpp!=4) || (ImageType==3&&Bpp!=1)){
        fclose(File);
        return false;
    }
    unsigned int ImageSize=*pw**ph*Bpp;
    unsigned char *data=(unsigned char *)malloc(ImageSize);
    // Читаем данные
    if(data==NULL||fread(data,1,ImageSize,File)!=ImageSize){
        free(data);
        fclose(File);
        return false;
    }
    // Закрываем файл
    fclose (File);
    *pdata=data;
    return true;
}
```

Пример использования функций записи и чтения данных из файлов формата Targa

```
int main(int argc, char **argv){
    if(argc!=3)
        return 1;
    unsigned char *data;
    unsigned w=0, h=0, bpp=0;
    //загрузка данных из файла, заданного в первом аргументе командной строки
    if(!Targa2Array(argv[1], &data, &w, &h, &bpp))
        return 1;

    //здесь производится обработка данных

    //запись данных в файл, заданного в первом аргументе командной строки
    Array2Image(argv[2], data2, w, h, bpp);
    //data надо освобождать вручную
    free(data);
    return 0;
}
```