МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт комплексной безопасности и цифровых технологий

Кафедра КБ-14 «Цифровые технологии обработки данных»

Платформы анализа больших данных

Лабораторная работа 7

Вариант 4

Выполнил:

Студент группы БСБО-09-22

Шутов Кирилл Сергеевич

Проверил:

Кашкин Евгений Владимирович

Москва, 2025

Постановка задачи

Применить детектор границ, вычисляя модуль градиента на изображении с помощью CUDA. Исходное изображение передаётся на вход программе, обрабатывается на графическом процессоре, и результат сохраняется в виде выходного изображения.

Описание кода и выполненных действий

С помощью библиотеки OpenCV входное изображение загружается и преобразуется в оттенки серого (grayscale), так как оператор Собеля применяется к одномерному яркостному каналу.

```
cv::Mat img = cv::imread(inPath, cv::IMREAD_GRAYSCALE);
```

Листинг 1. Загрузка и подготовка изображения

С помощью cudaMalloc на устройстве выделяется память под входное и выходное изображения.

```
cudaMalloc(&d_gray,  bytes);
cudaMalloc(&d_edges, bytes);
```

Листинг 2. Выделение памяти на GPU

Функция cudaMemcpy загружает данные изображения из оперативной памяти (host) на видеопамять (device).

```
cudaMemcpy(d_gray, img.data, bytes, cudaMemcpyHostToDevice);
```

Листинг 3. Копирование данных на устройство

На GPU запускается ядро sobelKernel, в котором для каждой точки изображения (за исключением границ) вычисляется горизонтальный и вертикальный градиент (Gx, Gy) с использованием масок оператора Собеля. Далее находится модуль градиента и результат сохраняется в выходной массив.

Обработанный массив с градиентами копируется обратно в оперативную память.

```
cudaMemcpy(h_edges.data(), d_edges, bytes, cudaMemcpyDeviceToHost);
```

Листинг 4. Получение результата

Финальное изображение с выделенными границами записывается в файл.

```
cv::imwrite(outPath, outImg);
```

Листинг 5. Сохранение изображения

Вывод

В результате выполнения лабораторной работы была реализована CUDA-программа, применяющая детектор границ на основе оператора Собеля.

Источники

1. Документация NVIDIA CUDA. [Электронный ресурс] URL: https://docs.nvidia.com/cuda/ Дата обращения: (12.05.2025 г).
2. OpenCV Documentation. [Электронный ресурс] URL: https://docs.opencv.org/ Дата обращения: (12.05.2025 г).
3. Wikipedia — Sobel operator. [Электронный ресурс] URL: https://en.wikipedia.org/wiki/Sobel_operator Дата обращения: (12.05.2025 г).

Приложение А. Листинг

```
#include <opencv2/opencv.hpp>
#include <cuda_runtime.h>
#include <iostream>

__global__ void sobelKernel(
	const unsigned char* __restrict__ gray,
	unsigned char* __restrict__ edges,
	int width, int height
) {
	int x = blockIdx.x * blockDim.x + threadIdx.x;
	int y = blockIdx.y * blockDim.y + threadIdx.y;
	if (x < 1 || x >= width - 1 || y < 1 || y >= height - 1) return;

	int idx = y * width + x;

	int gx =
		-gray[(y - 1) * width + (x - 1)] + gray[(y - 1) * width + (x + 1)]
		- 2 * gray[y * width + (x - 1)] + 2 * gray[y * width + (x + 1)]
		- gray[(y + 1) * width + (x - 1)] + gray[(y + 1) * width + (x + 1)];
	int gy =
		-gray[(y - 1) * width + (x - 1)] - 2 * gray[(y - 1) * width + x] -
gray[(y - 1) * width + (x + 1)]
		+ gray[(y + 1) * width + (x - 1)] + 2 * gray[(y + 1) * width + x] +
gray[(y + 1) * width + (x + 1)];

	int mag = sqrtf(float(gx * gx + gy * gy));
	if (mag > 255) mag = 255;
	edges[idx] = static_cast<unsigned char>(mag);
}

int main(int argc, char** argv) {
	if (argc != 3) {
		std::cerr << "Usage: " << argv[0] << " <input_image>
<output_image>\n";
		return 1;
	}
	const char* inPath = argv[1];
	const char* outPath = argv[2];

	cv::Mat img = cv::imread(inPath, cv::IMREAD_GRAYSCALE);
	if (img.empty()) {
		std::cerr << "Can't load image: " << inPath << "\n";
		return 1;
	}
	int width = img.cols;
	int height = img.rows;
	size_t numPixels = width * height;
	size_t bytes = numPixels * sizeof(unsigned char);

	unsigned char* d_gray = nullptr, * d_edges = nullptr;
	cudaMalloc(&d_gray, bytes);
	cudaMalloc(&d_edges, bytes);

	cudaMemcpy(d_gray, img.data, bytes, cudaMemcpyHostToDevice);

	dim3 block(16, 16);
	dim3 grid((width + block.x - 1) / block.x,
		(height + block.y - 1) / block.y);
	sobelKernel << <grid, block >> > (d_gray, d_edges, width, height);
	cudaDeviceSynchronize();

	std::vector<unsigned char> h_edges(numPixels);
	cudaMemcpy(h_edges.data(), d_edges, bytes, cudaMemcpyDeviceToHost);
```

```
        cv::Mat outImg(height, width, CV_8UC1, h_edges.data());
        cv::imwrite(outPath, outImg);

        cudaFree(d_gray);
        cudaFree(d_edges);

        return 0;
}
```