UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Part -4: Sobel Edge Detection using CUDA

Student Id	: 2407710
Student Name	: Dhadkan K.C.
Group	: L6CG4
Lecturer	: Mr.Yamu Poudel

Table of Contents

Overview 1
Header Files Used 1
Code Structure Overview 1
Task 4: Sobel Edge Detection Implementation 2
Operation 1: Reading Image File 2
Operation 2: Dynamic Memory Allocation 2
Operation 3: Sobel Edge Detection Algorithm 3
Operation 4: CUDA Multithreading Implementation 4
Operation 5: Output Image Generation 4
Compilation Command and Output 5
Compilation: 5
Run: 5
Input images: 5
Output images : 6
Conclusion 7

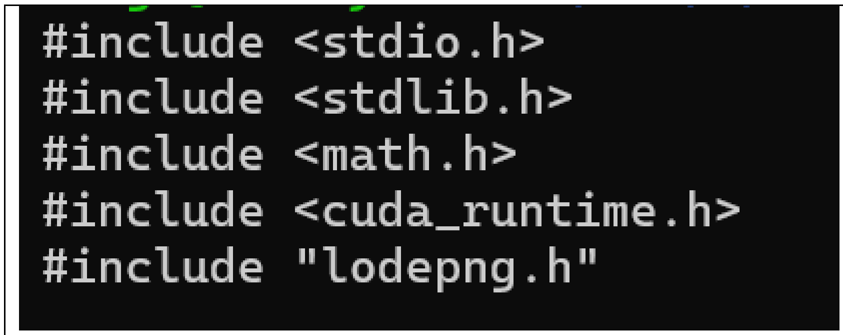
Tabe of Figures

Figure 1 Headerfile 1
Figure 2 Reading image 2
Figure 3 memory allocation 2
Figure 4 Sobel Edge Detection 3
Figure 5 CUDA multithreading 4
Figure 6 Output generation 4
Figure 7 lili.png 5
Figure 8 lotus.png 6
Figure 9 edges_lily 6
Figure 10 edge_lotus 7

Overview

This assignment uses Sobel edge detection based on CUDA to show image processing on the GPUs in parallel. The application takes one or more PNG files and converts them into grayscale, then executes the Sobel algorithm on the graphics card in parallel and with horizontal and vertical gradient kernels, and produces a single output image with the edges detected one each input image. The implementation brings out CUDA ideas including kernel execution, thread-level parallelism, and memory allocation in a GPU and memory transfers between the host and the device as well as appropriate edge detection and correct memory management.

Header Files Used



```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <cuda_runtime.h>
#include "lodepng.h"
```

Figure 1 Headerfile

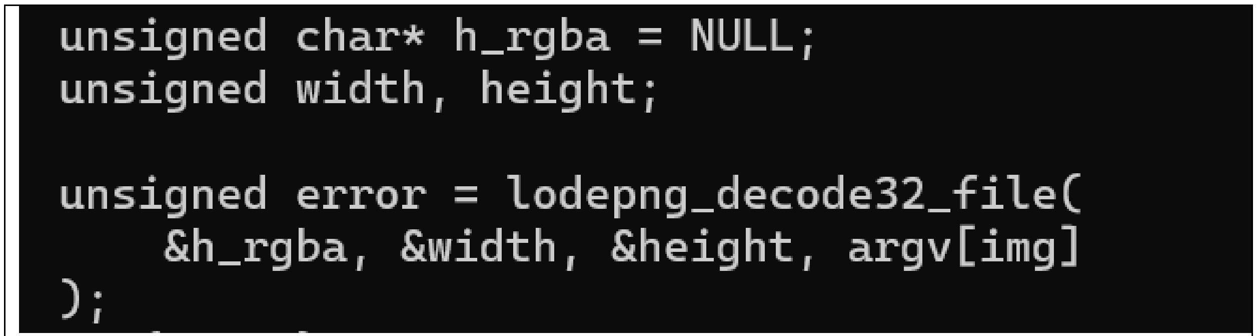
- **stdio.h**: Used for console input and output operations.
- **stdlib.h**: Provides dynamic memory allocation functions such as malloc() and free().
- **math.h**: Used for mathematical functions, specifically sqrtf() for gradient magnitude calculation.
- **cuda_runtime.h**: Required for CUDA runtime APIs including memory management and kernel execution.
- **lodepng.h**: Used to decode and encode PNG images without external dependencies.

Code Structure Overview

The program consists of two major parts including the GPU code and CPU code. The GPU code is made of a single CUDA kernel which is the Sobel edge detector where each thread is handling one pixel. The CPU code is used to read image files, convert image to grayscale, allocate and free memory, launch the CUDA kernel and write the output images to disk. This division of duties enhances better readability and maintainability and shows clearly the interaction between the host and the device.

Task 4: Sobel Edge Detection Implementation

Operation 1: Reading Image File

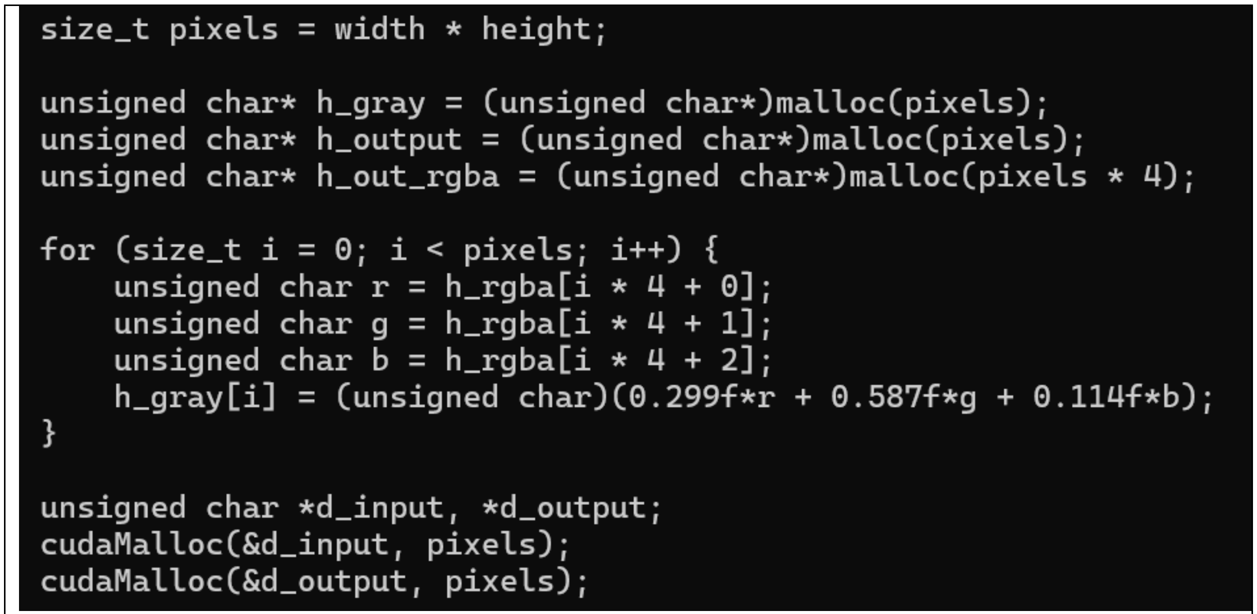


```
unsigned char* h_rgba = NULL;
unsigned width, height;

unsigned error = lodepng_decode32_file(
    &h_rgba, &width, &height, argv[img]
);
```

Figure 2 Reading image

The input PNG images are read using the lodepng library. Images are supplied through command-line arguments, allowing multiple images to be processed in one execution. Each image is decoded into RGBA format and stored in CPU memory.

Operation 2: Dynamic Memory Allocation

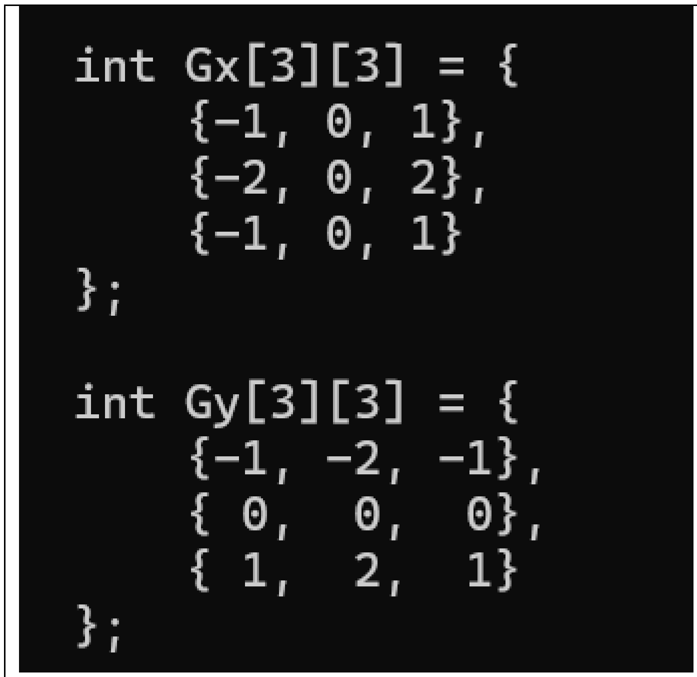


```
size_t pixels = width * height;

unsigned char* h_gray = (unsigned char*)malloc(pixels);
unsigned char* h_output = (unsigned char*)malloc(pixels);
unsigned char* h_out_rgba = (unsigned char*)malloc(pixels * 4);

for (size_t i = 0; i < pixels; i++) {
    unsigned char r = h_rgba[i * 4 + 0];
    unsigned char g = h_rgba[i * 4 + 1];
    unsigned char b = h_rgba[i * 4 + 2];
    h_gray[i] = (unsigned char)(0.299f*r + 0.587f*g + 0.114f*b);
}

unsigned char *d_input, *d_output;
cudaMalloc(&d_input, pixels);
cudaMalloc(&d_output, pixels);
```

Figure 3 memory allocation

The host and the device use dynamic memory allocation. The grayscale input and output images are stored in host memory and the input and output arrays used in GPU computation are stored in the device memory. The picture size is used to allocate memory so as to use it efficiently.

Operation 3: Sobel Edge Detection Algorithm

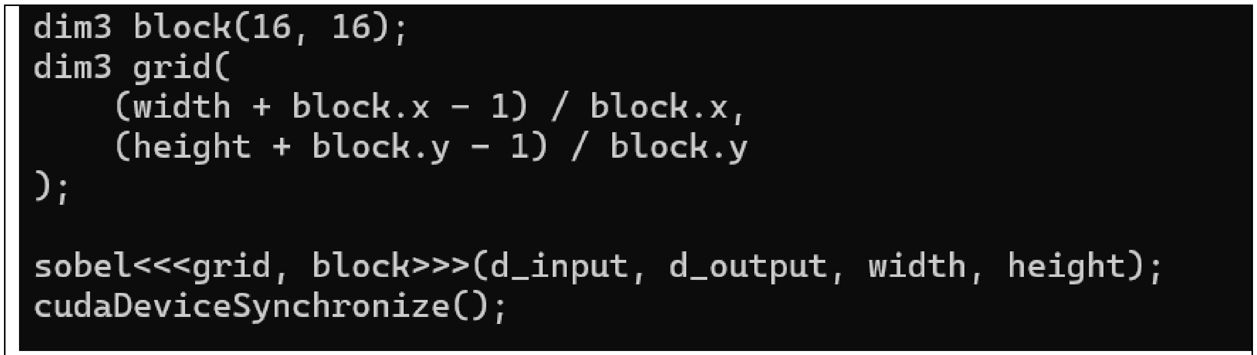


```
int Gx[3][3] = {
    {-1, 0, 1},
    {-2, 0, 2},
    {-1, 0, 1}
};

int Gy[3][3] = {
    {-1, -2, -1},
    { 0,  0,  0},
    { 1,  2,  1}
};
```

Figure 4 Sobel Edge Detection

The Sobel algorithm involves the use of two 3x3 convolution kernels. Gx and Gy identify horizontal and vertical intensity change respectively. Convolution is then performed on a neighborhood size of 3x3 on each pixel with a pad at the edges of the image. The strength of the final edge is obtained as the magnitude of the gradient formula.

Operation 4: CUDA Multithreading Implementation

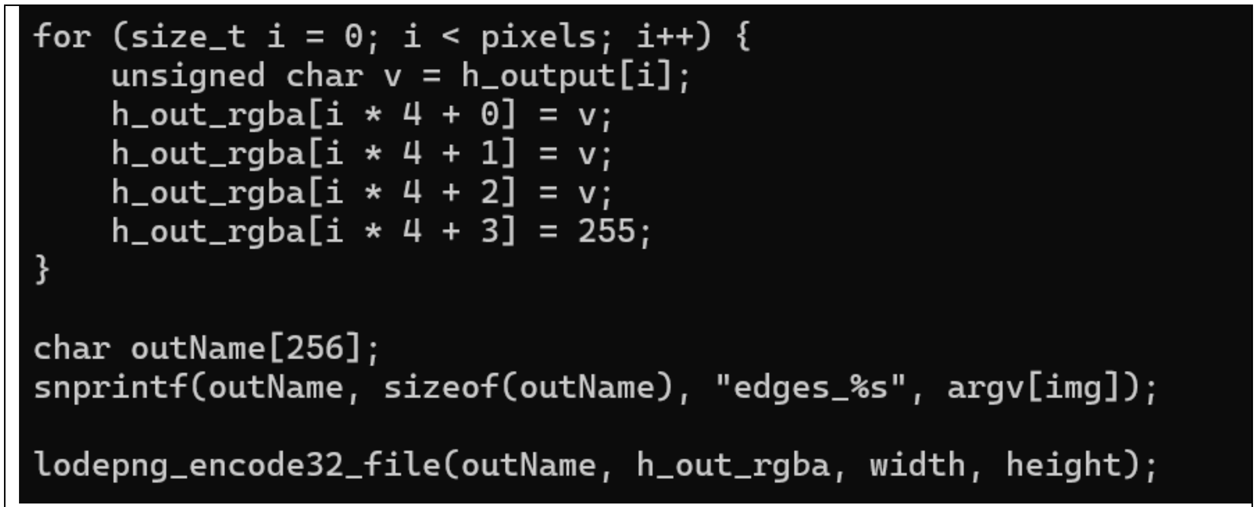


```
dim3 block(16, 16);
dim3 grid(
    (width + block.x - 1) / block.x,
    (height + block.y - 1) / block.y
);

sobel<<<grid, block>>>(d_input, d_output, width, height);
cudaDeviceSynchronize();
```

Figure 5 CUDA multithreading

CUDA multithreading is achieved by allocating one thread of the gpu per pixel. Two-dimensional grid and block structures enable processing of all the pixels in parallel. The magnitude of the gradient of the Sobel is individually calculated by each thread on the respective pixel.

Operation 5: Output Image Generation



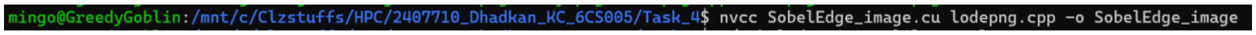
```
for (size_t i = 0; i < pixels; i++) {
    unsigned char v = h_output[i];
    h_out_rgba[i * 4 + 0] = v;
    h_out_rgba[i * 4 + 1] = v;
    h_out_rgba[i * 4 + 2] = v;
    h_out_rgba[i * 4 + 3] = 255;
}

char outName[256];
snprintf(outName, sizeof(outName), "edges_%s", argv[img]);

lodepng_encode32_file(outName, h_out_rgba, width, height);
```

Figure 6 Output generation

Once the image has been processed with GPU, the image with the edges identified is transferred back to the memory of the CPU, converted to RGBA and saved to a PNG file. Output file is marked with edges_ to differentiate the file with the original image.

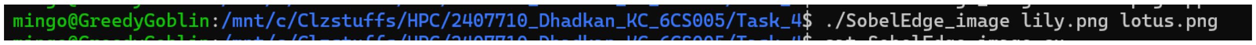
Compilation Command and Output

Compilation:



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/2407710_Dhadkan_KC_6CS005/Task_4$ nvcc SobelEdge_image.cu lodepng.cpp -o SobelEdge_image
```

Run:



```
mingo@GreedyGoblin:/mnt/c/Clzstuffs/HPC/2407710_Dhadkan_KC_6CS005/Task_4$ ./SobelEdge_image lily.png lotus.png
```

Input images:

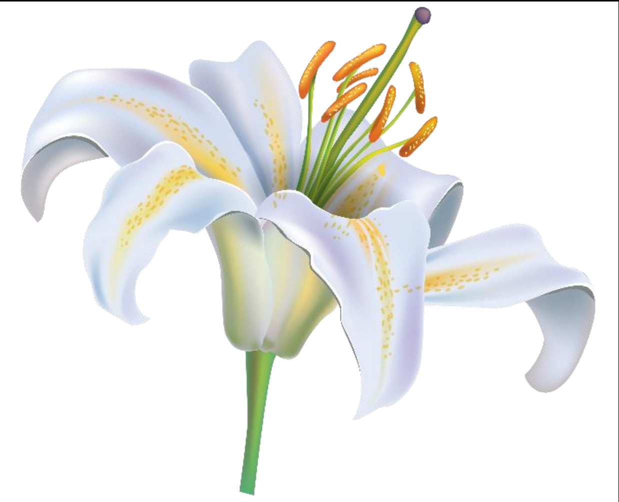


Figure 7 lili.png

Figure 8 lotus.png

Output images :

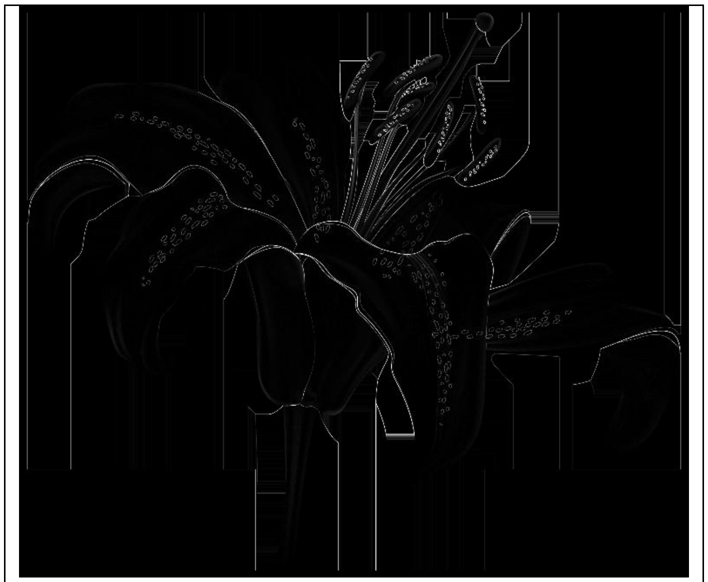


Figure 9 edges_lily

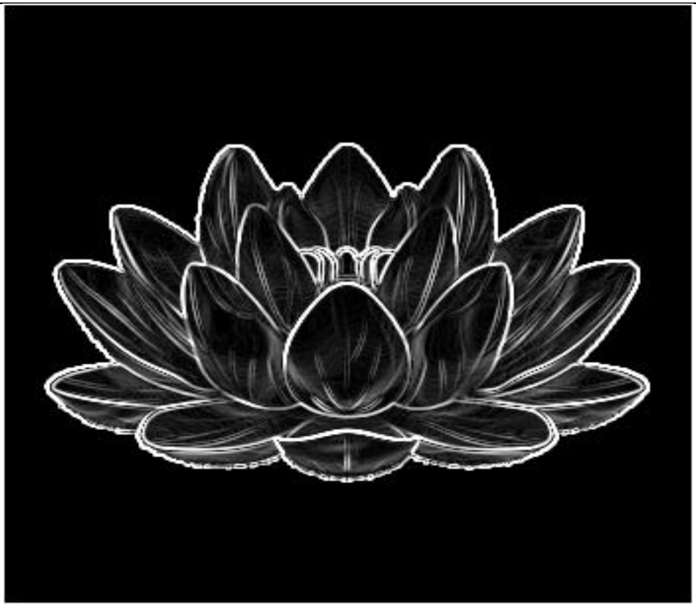


Figure 10 edge_lotus

Limitations

- Colour edge detection is not provided, though Sobel is applied to grayscale images only.
- The Sobel kernel is fixed at 3x3 and bigger or adaptive kernels are not available.
- Memory optimisations are not shared, which can be enhanced to improve performance.
- CIUD API error detection is small. The program only uses the lodepng library to support PNG images.

Conclusion

This exercise is able to show how Sobel edge detector can be used with CUDA. The program is able to process multiple images simultaneously, properly apply convolution with Gx and Gy kernels, handle computer and graphics memory safely and give correct output images with edges detected. The implementation satisfies the requirements of all the tasks and provides a good example of the utilization of the image processing acceleration by using the GPU.