Sobel filter na CUDA – Ubrzanje obrade slike pomoću paralelizma

Ivan Akrapović, Jozo Krstanović, Dominik Mišadin

Split, lipanj 2025.

Uvod

- Detekcija rubova ključna u obradi slike
- CPU tradicionalno korišten, ali ograničen za real-time primjene
- GPU omogućuje paralelnu obradu milijuna piksela

Slika 3.2. Originalna slika (lijevo) i detektirani rubovi (desno)

Što je CUDA?

- Razvojna platforma za paralelno programiranje na GPU-u
- Omogućuje pisanje C/C++ koda koji se izvršava na GPU-u
- Ključni pojmovi: __global__, threadIdx, blockIdx, gridDim, blockDim
- Pogodna za obradu slike zbog neovisne obrade svakog piksela

Sobel filter

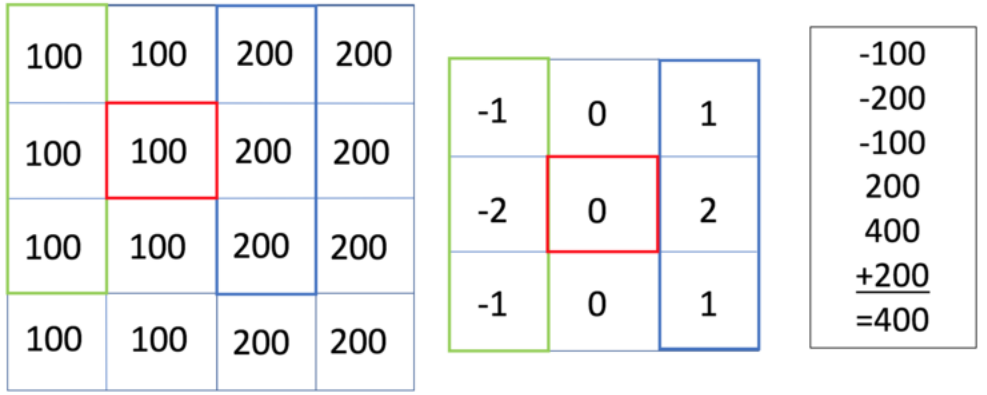
- Algoritam za detekciju rubova
- Koristi horizontalni (G_X) i vertikalni (G_Y) 3x3 kernel
- Računa gradijent intenziteta ⟶ označava "jačinu ruba"
- Formula: $magnitude = \sqrt{g_x^2 + g_y^2}$

X – Direction Kernel

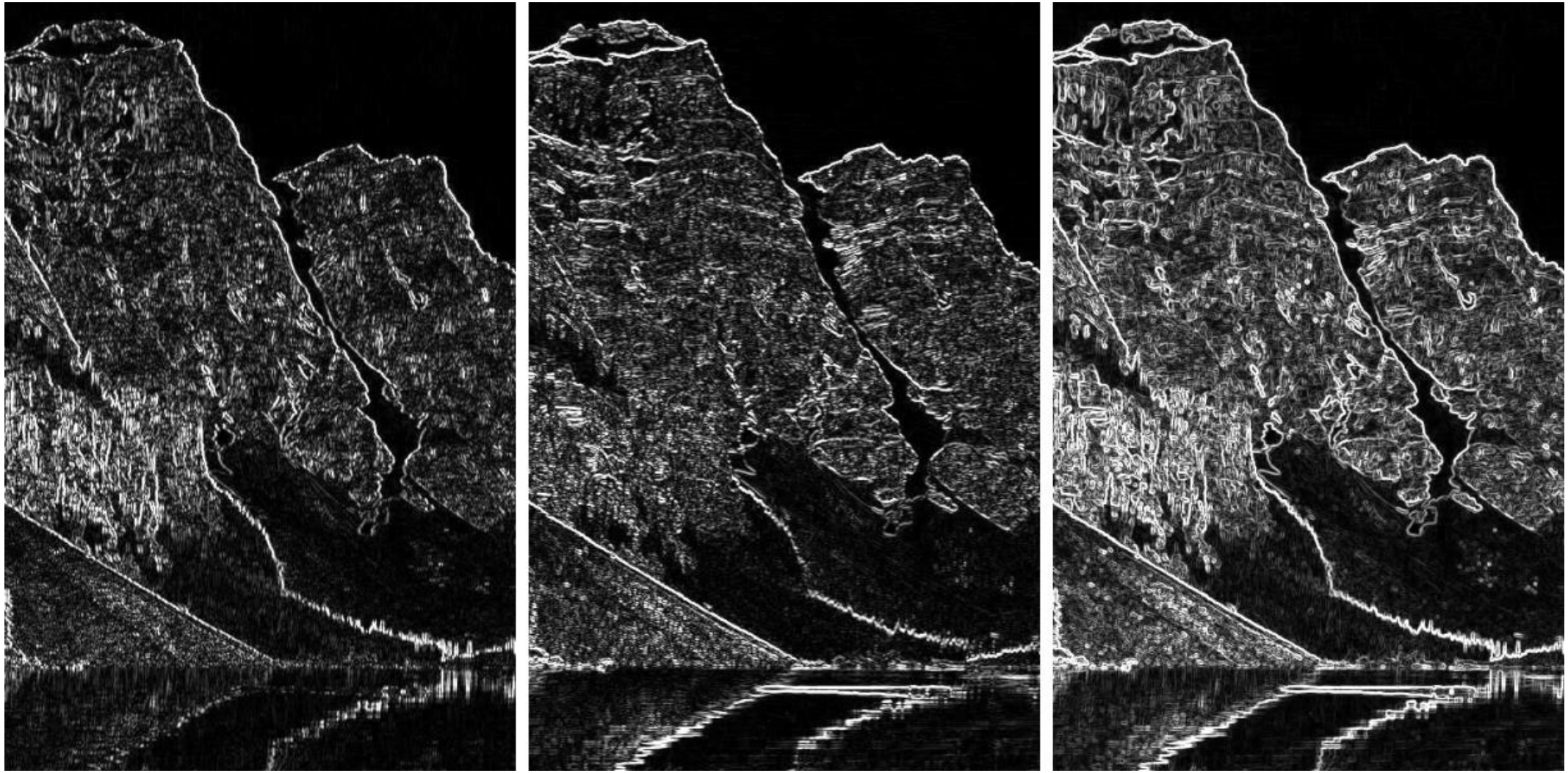
-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1



Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.



Slika 3.2. Usporedba prolaza X-osi (lijevo) i Y-osi (sredina) te prolaz sa spojenom magnitudom (desno)

CPU implementacija

- Implementirano u C++ pomoću OpenCV
- Petlje kroz piksele, ručni izračun G_X, G_Y i magnitude
- Vrijeme izvođenja:
  - 32 ms (C++),
  - 138 ms (Python)

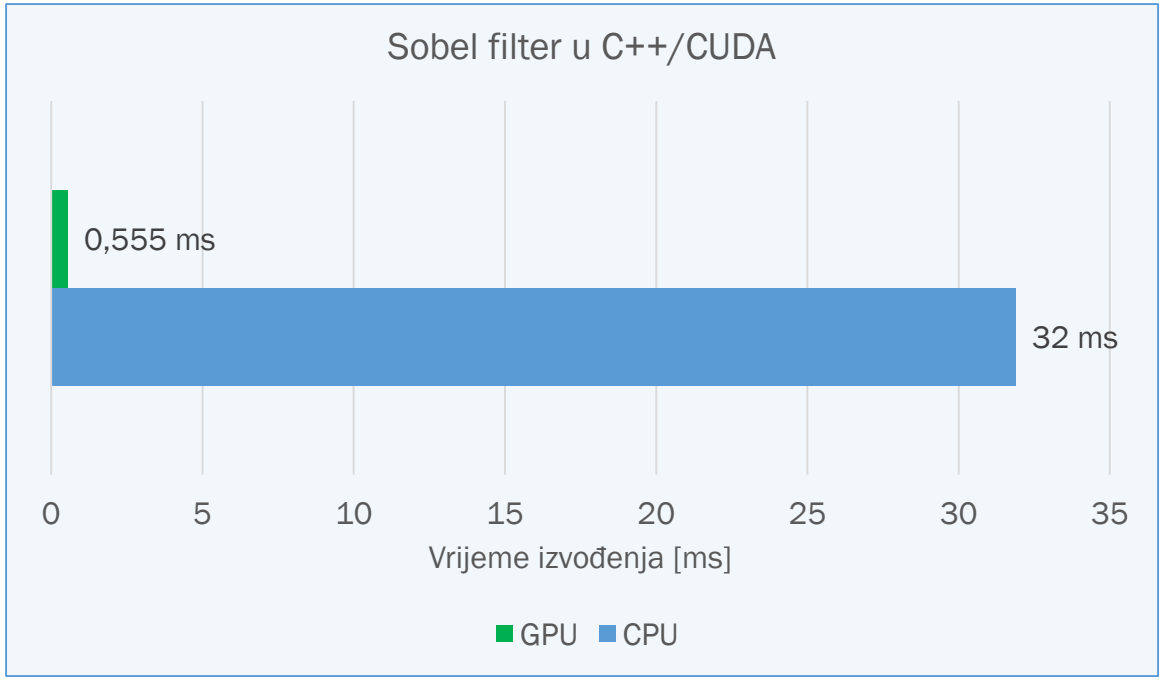
```
void sobelFilterCPU(const cv::Mat& gray, cv::Mat& output) {
    int width = gray.cols;
    int height = gray.rows;

    output = cv::Mat::zeros(height, width, CV_8U);

    for (int y = 1; y < height - 1; ++y) {
        for (int x = 1; x < width - 1; ++x) {
            int gx = - gray.at<uchar>(y - 1, x - 1)
                     - 2 * gray.at<uchar>(y, x - 1)
                     - gray.at<uchar>(y + 1, x - 1)
                     + gray.at<uchar>(y - 1, x + 1)
                     + 2 * gray.at<uchar>(y, x + 1)
                     + gray.at<uchar>(y + 1, x + 1);

            int gy = - gray.at<uchar>(y - 1, x - 1)
                     - 2 * gray.at<uchar>(y - 1, x)
                     - gray.at<uchar>(y - 1, x + 1)
                     + gray.at<uchar>(y + 1, x - 1)
                     + 2 * gray.at<uchar>(y + 1, x)
                     + gray.at<uchar>(y + 1, x + 1);

            int magnitude = sqrtf((float)(gx * gx + gy * gy));

            output.at<uchar>(y, x) = magnitude > 255 ? 255 : magnitude;
        }
    }
}
```

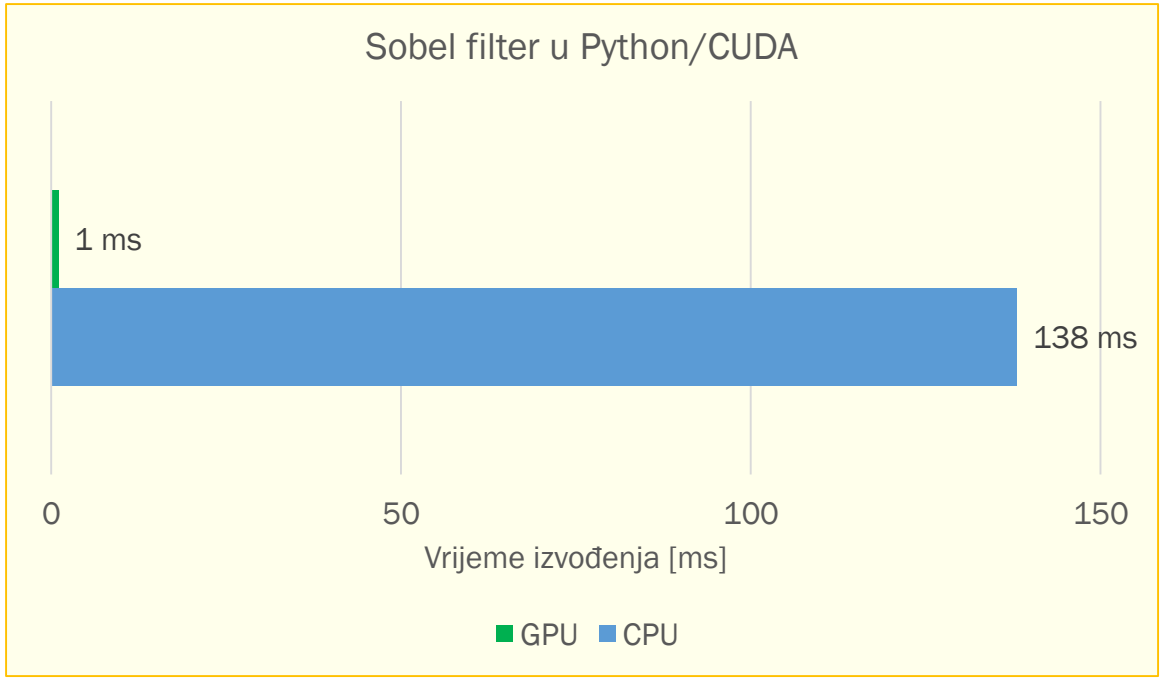
GPU implementacija

- Paralelno računanje nad svakim pikselom
- Dvije faze: pretvorba u sivu i Sobel filter
- Korištenje grid i block konfiguracije
- Vrijeme izvođenja:
  - 0.555 ms (C++),
  - 1 ms (Python)

```
__global__ void sobelFilter(const unsigned char* gray,
                            unsigned char* output, int width, int height)
{
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;

    if (x <= 0 || x >= width - 1 || y <= 0 || y >= height - 1) return;

    int idx = y * width + x;

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

    output[idx] = magnitude > 255 ? 255 : magnitude;
}
```

Usporedba rezultata





Zaključak

- GPU implementacija Sobel filtra pokazala je ubrzanje od 80 do 130 puta u odnosu na CPU.
- Vrijeme obrade slike (3000×2000 px):
  - CPU: 32 ms (C++), 138 ms (Python)
  - GPU: 0.555 ms (C++), 1 ms (Python)
- Vizualna kvaliteta rezultata između CPU i GPU verzija je gotovo identična.
- CUDA omogućuje efikasnu i brzu obradu slike u stvarnom vremenu.
- Idealno rješenje za primjene s visokim zahtjevima za brzinom:
  - video analitika, autonomna vožnja, industrijski sustavi nadzora
- Analiza potvrđuje veliki praktični doprinos paralelizacije u računalnom vidu.