멀티미디어 병렬프로그래밍 기말 프로젝트

고속도로 주행 영상에 대한 차선 검출 프로그램 병렬화

...

2014112553 김 태 윤
2013112668 이 호 민

INDEX

1. 프로젝트 주제
2. 알고리즘 & 결과물
3. 병렬화 부분
4. 이슈 및 추가기능

1. 프로젝트 주제 소개

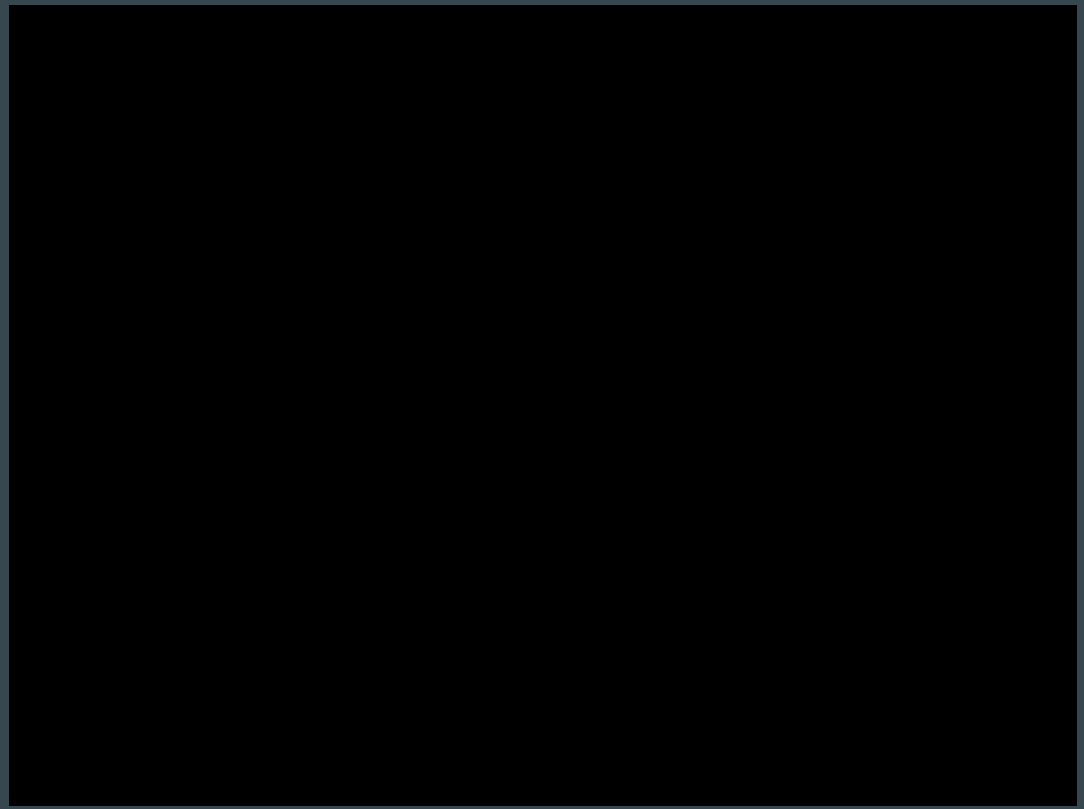
주제: 고속도로 주행 영상의 차선을 검출하고 검출한 차선에 대하여 선을 그리는 프로그램을 병렬 프로그래밍으로 처리한다.

사용하는 알고리즘: 허프 변환 알고리즘

2. 사용한 알고리즘

1. 컬러영상을 gray로 전환하고 Gaussian Blur를 적용
2. 전환된 영상을 Canny edge를 사용하여 변환시킨다.
3. 화면의 모든 점에 대해 허프 변환을 적용한다.
4. 적용된 결과를 선으로 표현한다.

2. 최종 결과물



3. 병렬화 부분

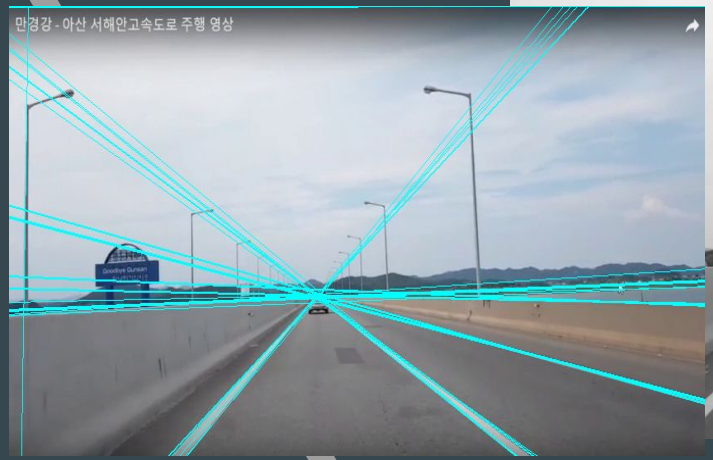
colorTogray: RGB영상을 Gray영상으로 변환

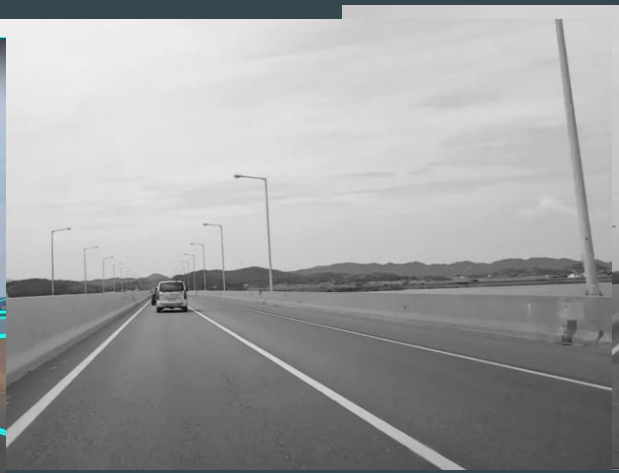
Gaussian Blur: 5x5마스크 행렬을 통해 약간의 blur처리

Hough Transform: 각 픽셀을 극좌표계로 전환하고,
교점들의 수를 통해 직선을 검출

BGR2GRAY & GaussianBlur: 허프변환을 위한 전처리

RGB영상을 Grayscale로 변환 & Canny검출을 위한 blurring





〈뚜렷한 성능 차이는 보이지 않음 〉

Gray Scale 변환 global함수 소스코드

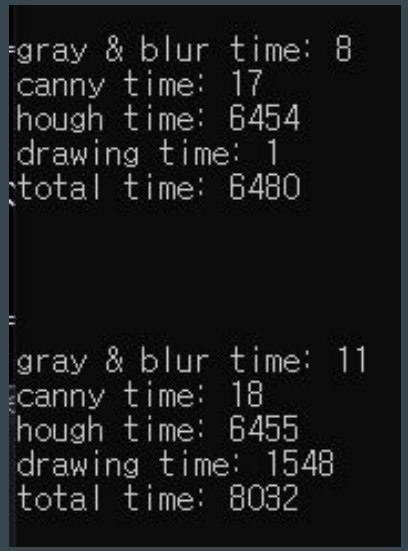
```
__global__ void rgb_2_grey(unsigned char *greyImage, uchar3 *color, int rows, int columns)
{
    int rgb_x = blockIdx.x * TILE_WIDTH + threadIdx.x;
    int rgb_y = blockIdx.y * TILE_WIDTH + threadIdx.y;

    if ((rgb_x < columns) && (rgb_y < rows)) {
        int rgb_ab = rgb_y*columns + rgb_x;
        uchar3 rgbImg = color[rgb_ab];
        greyImage[rgb_ab] = unsigned char((float(rgbImg.x))*0.299f + (float(rgbImg.y))*0.587f + (float(rgbImg.z))*0.114f);
    }
}
```

Gaussian Blur global함수 소스코드

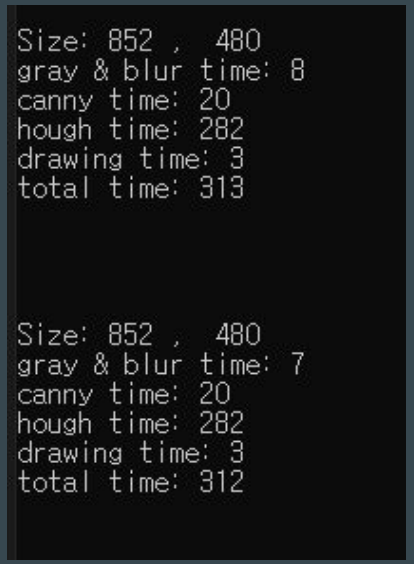
```
__global__ void GaussianMask(unsigned char *output, const unsigned char *src, int rows, int columns)
{
    float gaussianMask[] = { 0.0000, 0.0000, 0.0002, 0.0000, 0.0000,
        0.0000, 0.0113, 0.0837, 0.0113, 0.0000,
        0.0002, 0.0837, 0.6187, 0.0837, 0.0002,
        0.0000, 0.0113, 0.0837, 0.0113, 0.0000,
        0.0000, 0.0000, 0.0002, 0.0000, 0.0000 };

    int x = blockIdx.x * TILE_WIDTH + threadIdx.x;
    int y = blockIdx.y * TILE_WIDTH + threadIdx.y;

    if ((x > 2 || x < columns - 2) && (y > 2 || y < rows - 2))
    {
        float mask = 0;

        for (int r = 0; r < 5; r++)
        {
            for (int c = 0; c < 5; c++)
            {
                int idx = (y + r - 2) * WIDTH + x + c - 2;
                mask += gaussianMask[r * 5 + c] * src[idx];
            }
        }
            output[y*WIDTH + x] = (unsigned char)(mask * 0.9);

    }
    else
        output[y*WIDTH + x] = 0;

}
```

CPU와 GPU 결과 비교

CPU 결과 창



GPU 결과 창



Hough Transform: 허프 변환을 통한 선 검출

시도1. 2차원 Grid로 global함수 구현

각 픽셀을 극좌표로 전환하여 교차점에 투표(voting)

모든 픽셀 탐색시 약25배, 선 탐색 시 20배의 성능 향상

시도2. 3차원 Grid로 global함수 구현

2차원 쓰레드가 반복하던 연산을 3차원으로 해결

2차원 Grid보다 약 200ms의 성능 향상

2차원 Grid global함수의 결과 및 소스코드

```
Size: 852 ,  480
gray & blur time: 8
canny time: 18
hough time: 198
drawing time: 3
total time: 227


Size: 852 ,  480
gray & blur time: 7
canny time: 19
hough time: 198
drawing time: 2
total time: 226
```

결과 창

```
__global__ void HoughTransform2(unsigned char* src, int* houghspace, int centerX, int centerY, int diagonal)
{
    int i = blockIdx.y * 32 + threadIdx.y;
    int j = blockIdx.x * 32 + threadIdx.x;

    if (i < HEIGHT && j < WIDTH) {
        for (int angle = 0; angle < 180; angle++)
        {
            int r = (j - centerX) * cos(angle * CV_PI / 180) + (i - centerY) * sin(angle * CV_PI / 180);
            r += diagonal;
            if (src[i * WIDTH + j] > 0) {

                atomicAdd(&houghspace[r * 180 + angle], 1);
            }
        }
    }
}
```

소스 코드

3차원 Grid global함수의 결과 및 소스코드

```
Size: 852 ,  480
gray & blur time: 9
canny time: 26
hough time: 125
drawing time: 2
total time: 162



Size: 852 ,  480
gray & blur time: 8
canny time: 24
hough time: 122
drawing time: 2
total time: 156
```

결과 창

```cpp
__global__ void HoughTransform(unsigned char* src, int* houghspace, int centerX, int centerY, int diagonal)
{
    int i = blockIdx.y * 16 + threadIdx.y;
    int j = blockIdx.x * 16 + threadIdx.x;
    int k = blockIdx.z * 4 + threadIdx.z;
    if (i < HEIGHT && j < WIDTH) {
        if (src[i * WIDTH + j] > 0) {
            int temp = xy2rtheta(i, j, diagonal, centerX, centerY, k);
            atomicAdd(&houghspace[temp], 1);
        }
    }
}


__device__ int xy2rtheta(int i, int j, int diagonal, int centerX, int centerY, int angle)
{
    int r = (j - centerX) * cos(angle * CV_PI / 180) + (i - centerY) * sin(angle * CV_PI / 180);
    r += diagonal;
    return r * 180 + angle;
}
```

소스 코드

5. 구현 이슈 및 추가기능

1. **교차점 갯수 연산에 증감연산자 사용**

   Atomic연산자를 이용하여 에러를 방지

2. **키 입력으로 3가지 필터효과**

   1번 차선검출, 2번 grayscale, 3번 gaussian, 4번 canny

감사합니다