FilteringView.cpp

```
    void GaussianFilteringGPU(int flag){
                                  :
          Do_Gaussian_on_GPU( ...   , flag)
```

$\swarrow$

cuda_code.cu

```
float Do_Gaussian_on_GPU(          ){
   Set_Gaussian_kernel();

   if(flag=SHARED){
      Gaussian_kernel_shared(p_bitmaps, p_Gaussian,
                                    width, height);
   }
   else {
     Gaussian_kernel_no_shared(p_bitmaps, p_Gaussian,
                                    width, height);
}
```

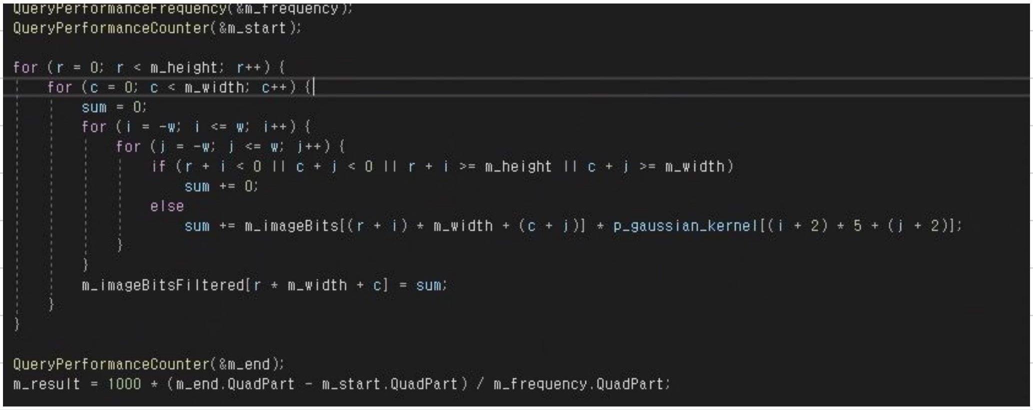
```
__global__ void Gaussian_kenel_shared (INOUT unsigned char * d_bitmaps
                                       OUT unsigned char * d_Gaussian
                                       long width, long height)
{ int row = blockIdx.y * blockDim.y + threadIdx.y;
  int col = blockIdx.x * blockDim.x + threadIdx.x;
  int id = width * row + col; // image id
  int s_id = threadIdx.y * blockDim.y + threadIdx.x   // sharedmemory_id
  // 범위밖 → 예외처리
  if(row >= height // col >= width) return;
  // 초기화
  d_Gaussian[id] = 0;


  if (row < height && col < width) {
     sharedBuffer[threadIdx.y * blockDim.x + threadIdx.x]
          = d_bitmaps [row * width + col]; // input된이미지 파일을
                                              shared memory에 저장
  }

 __syncthreads();
 if (threadIdx.x >= 5) && (threadIdx.x < blockDim.x - 5)
     && (threadIdx.y >= 5 ) && ( threadIdx.y < blockDim.y - 5))       범위내경우만고려
 {
    int sum = 0;                                                          kernel이 5X5이므로
    for(int dy = -5; dy < 5; dy++) {                                      한 픽셀에 5X5=25번의
     for(int dx = -5; dx < 5; dx++) {                                     연산 수행.
        int i = sharedBuffer [s_id + (dy * blockDim.y) + dx];
        sum += constant_gaussian_kernel [dy * 5 + dx] * i ;
     }
   }
  d_Gaussian[id] = sum;
}
```

```
__global__ void Gaussian_kernel_no_shared (IN unsigned char *d_bitmaps,
                                           OUT unsigned char *d_Gaussian,
                                           long width, long height) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    int sum = 0;
    if (row <= 0 || row >= height-1 || col <= 0 || col >= width-1)
        d_Gaussian[row*width + col] = 0;                    범위를 벗어날
                                                              경우
                                                            예외처리
    else {
        for(int i=-2; i<=2; i++) {
            for(int j=-2; j<=2; j++) {
                int k = d_bitmaps[(row+i)*width + (col+j)];
                sum += k * constant_Gaussian_kernel[i+2][j+2];
            }                                       kernel의 한가운데를
        }                                           중심으로 인덱스조절함.
    d_Gaussian [row*width + col] = sum;
}
```

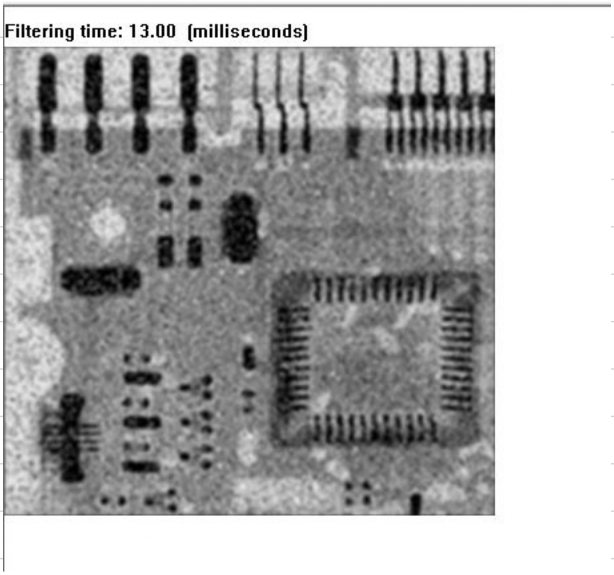
CPU



```
QueryPerformanceFrequency(&m_frequency);
QueryPerformanceCounter(&m_start);

for (r = 0; r < m_height; r++) {
    for (c = 0; c < m_width; c++) {
        sum = 0;
        for (i = -w; i <= w; i++) {
            for (j = -w; j <= w; j++) {
                if (r + i < 0 || c + j < 0 || r + i >= m_height || c + j >= m_width)
                    sum += 0;
                else
                    sum += m_imageBits[(r + i) * m_width + (c + j)] * p_gaussian_kernel[(i + 2) * 5 + (j + 2)];
            }
        }
        m_imageBitsFiltered[r * m_width + c] = sum;
    }
}

QueryPerformanceCounter(&m_end);
m_result = 1000 * (m_end.QuadPart - m_start.QuadPart) / m_frequency.QuadPart;
```



Filtering time: 13.00 (milliseconds)