// erode filter

\_\_global\_\_ void Morphology\_erode\_kernel(

unsigned char\* input,

unsigned char\* ouput,

int nWidth,

int nHeight,

int nWdithStep) {

const int ix = blockIdx.x\*blockDim.x + threadIdx.x;

const int iy = blockIdx.y\*blockDim.y + threadIdx.y;

const int ix\_1 = max(0, ix - 1);

const int ix1 = min(nWidth - 1, ix + 1);

const int iy\_1 = max(0, iy - 1);

const int iy1 = min(nHeight - 1, iy + 1);

if (ix < nWidth&&iy < nHeight) {

if (input[iy\*nWdithStep + ix]) {

if (input[iy\_1\*nWdithStep + ix\_1] == 0 ||

input[iy\_1\*nWdithStep + ix] == 0 ||

input[iy\_1\*nWdithStep + ix1] == 0 ||

input[iy\*nWdithStep + ix\_1] == 0 ||

input[iy\*nWdithStep + ix1] == 0 ||

input[iy1\*nWdithStep + ix\_1] == 0 ||

input[iy1\*nWdithStep + ix] == 0 ||

input[iy1\*nWdithStep + ix1] == 0

) {

ouput[iy\*nWdithStep + ix] = 0;

}

}

}

}

// 均值滤波

\_\_global\_\_ void ImgFilter\_3x3\_m\_Kernel(

unsigned char \*input,

unsigned char \*output,

int nWidth,

int nHeight,

int nWdithStep

) {

const int ix = blockIdx.x\*blockDim.x + threadIdx.x;

const int iy = blockIdx.y\*blockDim.y + threadIdx.y;

const int ix\_1 = max(0, ix - 1);

const int ix1 = min(nWidth - 1, ix + 1);

const int iy\_1 = max(0, iy - 1);

const int iy1 = min(nHeight - 1, iy + 1);

if (ix < nWidth&&iy < nHeight) {

int nTemp;

nTemp = input[iy\_1\*nWdithStep + ix\_1];

nTemp += input[iy\_1\*nWdithStep + ix];

nTemp += input[iy\_1\*nWdithStep + ix1];

nTemp += input[iy\*nWdithStep + ix\_1];

nTemp += input[iy\*nWdithStep + ix];

nTemp += input[iy\*nWdithStep + ix1];

nTemp += input[iy1\*nWdithStep + ix\_1];

nTemp += input[iy1\*nWdithStep + ix];

nTemp += input[iy1\*nWdithStep + ix1];

nTemp /= 9;

output[iy\_1\*nWdithStep + ix] = nTemp;

}

}

// 四邻域二姐边缘算子

/\*

0 1 0

1 -4 1

0 1 0

\*/

//Laplacian算子边缘检测

\_\_global\_\_ void ImgFilter\_3x3\_Kernel(

unsigned char \*input,

unsigned char \*output,

int nWidth,

int nHeight,

int nWdithStep

) {

const int ix = blockIdx.x\*blockDim.x + threadIdx.x;

const int iy = blockIdx.y\*blockDim.y + threadIdx.y;

const int ix\_1 = max(0, ix - 1);

const int ix1 = min(nWidth - 1, ix + 1);

const int iy\_1 = max(0, iy - 1);

const int iy1 = min(nHeight - 1, iy + 1);

if (ix < nWidth&&iy < nHeight) {

int nTemp;

nTemp = input[iy\_1\*nWdithStep + ix];

nTemp += input[iy\*nWdithStep + ix\_1];

nTemp -= input[iy\*nWdithStep + ix]<<2;

nTemp += input[iy\*nWdithStep + ix1];

nTemp += input[iy1\*nWdithStep + ix1];

nTemp =abs(nTemp);

nTemp =min(255,nTemp);

output[iy\*nWdithStep + ix] = nTemp;

}

}