



PROJET CUDA

Traitement d'images

ZHANG Ling

STEDE-SCHRADER Anthony

2020 - 2021

SOMMAIRE

0 - Compilation et exécution	3
1 - Redimensionner une image	3
2 - Rotation d'une image	5
3 - Flou Gaussien et débruitage	6
4 - Détection Sobel	9
5 - Histogramme d'une image	11

0 - Compilation et exécution

Pour compiler et exécuter le programme et ses différentes sous-parties, les instructions sont indiquées dans le README.md.

1 - Redimensionner une image

Fichier : Zoomer.cu

Explication : On effectue zoom pour l'image (si on veut dézoomer, juste changer la fonction kernel)

```
__global__ void kernel(uchar* _src_dev, uchar* _dst_dev, int _src_step, int _dst_step,
int _src_rows, int _src_cols, int _dst_rows, int _dst_cols)
{
    auto i = blockIdx.x;
    auto j = blockIdx.y;

    double fRows = _dst_rows / (float)_src_rows;
    double fCols = _dst_cols / (float)_src_cols;

    auto pX = 0;
    auto pY = 0;

    pX = (int)(i / fRows);
    pY = (int)(j / fCols);
    if (pX < _src_rows && pX >= 0 && pY < _src_cols && pY >= 0) {
        *(_dst_dev + i * _dst_step + 3 * j + 0) = *(_src_dev + pX * _src_step + 3 * pY);
        *(_dst_dev + i * _dst_step + 3 * j + 1) = *(_src_dev + pX * _src_step + 3 * pY + 1);
        *(_dst_dev + i * _dst_step + 3 * j + 2) = *(_src_dev + pX * _src_step + 3 * pY + 2);
    }
}
```

Problèmes rencontrés : Une petite exception avec CV_DbgAssert(p == buf) (mat.inl.hpp) quand il compile, mais on pourra l'ignorer.

```
inline MatStep::operator size_t() const
{
    CV_DbgAssert( p == buf );
    return buf[0];
}

inline MatStep& MatStep::operator= (const MatStep& s)
{
    CV_DbgAssert( p == buf );
    buf[0] = s;
    return *this;
}
```

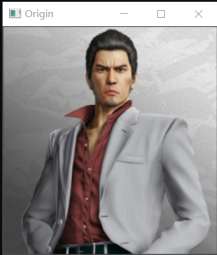
Exception Unhandled
Unhandled exception at 0x00007FFAB2504B59 in CUDA 11.2
Runtime1.exe: Microsoft C++ exception: cv::Exception at memory location 0x0000005B84CFF070.
[Copy Details](#)
[Exception Settings](#)

Résultats :


C:\Users\ginto\Desktop\projet_cuda_zhang_stedeschrader\CUDA 11.2 Runtime1\64\Debug\CUDA 11.2 Runtime1.exe

cpu cost time: 110
OpenCV(4.5.1) Error: Assertion failed (p == buf) in cv::MatStep::operator unsigned long long, file D:\openCV\opencv\buil
d\include\opencv2\core/mat.inl.hpp, line 1197
OpenCV(4.5.1) Error: Assertion failed (p == buf) in cv::MatStep::operator unsigned long long, file D:\openCV\opencv\buil
d\include\opencv2\core/mat.inl.hpp, line 1197
gpu cost time: 137813

Origin



Zoom



typename _Tp inline
cv::Mat(int rows, int cols)

Value	Type
0x0000005b84cf4f80 (804, 3)	unsigned_
0x0000005b84cf630 (804)	unsigned_
0x0000005b84cf478 (p=0x0000005b84cf630, cv::MatStep	

2 - Rotation d'une image

Fichier : Rotate.cu

Explication : La fonction `rotatImage()` effectue une rotation à 90° sur la droite de l'image envoyée sur le CPU. Pour cet algorithme, on inverse juste les colonnes et les lignes de pixels pour effectuer la transformation. Pour la partie GPU, on utilise la fonction `rotatImageGpu()`. Pour cette fonction, on a utilisé `cv::cuda::rotate` afin d'effectuer la transformation de l'image.

Problèmes rencontrés : Ça ne fonctionne malheureusement pas, et on n'a pas réussi à savoir pourquoi. Le problème viendrait de la fonction `cv::cuda::rotate`, mais nous avons trouvé des exemples qui utilisaient ce principe d'une façon assez similaire à la nôtre.

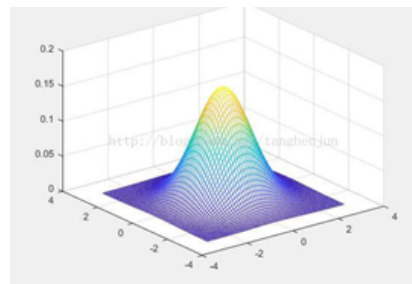
Résultats : N/A

3 - Flou Gaussien et débruitage

Fichier : Gaussian_and_denoising(bilateral_filtering).cu

Explication :

Pour la partie Gaussien, l'idée est d'utiliser un modèle (ou masque de convolution) pour scan chaque pixel de l'image et utilise la valeur de gris moyenne pondérée des pixels dans le voisinage déterminé par le modèle pour remplacer la valeur du pixel central du modèle.



La méthode de filtrage gaussienne que j'ai utilisée est celle illustrée dans la figure ci-dessus. Après avoir déterminé le kernel gaussien, on ignore la taille du 1/2 kernel du bord pour s'assurer que l'opération de convolution peut s'exécuter correctement. Une fois que la valeur d'entrée est multipliée par le coefficient de kernel gaussien, on peut déterminer si le résultat dépasse la valeur gris, et le traiter.

```
//Gaussian filtering
__global__ void GaussianFilterInCuda(unsigned char* dataIn, unsigned char* dataOut, cv::Size erodeElement, int imgWidth, int imgHeight)
{
    int xIndex = threadIdx.x + blockIdx.x * blockDim.x;
    int yIndex = threadIdx.y + blockIdx.y * blockDim.y;

    int Index = xIndex + yIndex * imgWidth;

    int elementWidth = erodeElement.width;
    int elementHeight = erodeElement.height;
    int halfEW = elementWidth / 2;
    int halfEH = elementHeight / 2;

    //Initialization output
    dataOut[Index] = dataIn[Index];

    //Prevent cross-border halfEW < xIndex < imgWidth-halfEW halfEH < yIndex < imgHeight-halfEH
    if (xIndex > halfEW && xIndex < imgWidth - halfEW && yIndex > halfEH && yIndex < imgHeight - halfEH)
    {
        int sum = 0;
        for (int i = -halfEW; i < halfEW + 1; i++)
        {
            for (int j = -halfEH; j < halfEH + 1; j++)
            {
                /* if (dataIn[(i + yIndex) * imgWidth + xIndex + j] < dataOut[yIndex * imgWidth + xIndex])
                {
                    dataOut[yIndex * imgWidth + xIndex] = dataIn[(i + yIndex) * imgWidth + xIndex + j];
                }*/

                sum += dataIn[(i + yIndex) * imgWidth + xIndex + j] * d_const_Gaussian[(i + 2) * 5 + j + 2];
            }
        }

        if (sum / 273 < 0)
            dataOut[yIndex * imgWidth + xIndex] = 0;
        else if (sum / 273 > 255)
            dataOut[yIndex * imgWidth + xIndex] = 255;
        else
            dataOut[yIndex * imgWidth + xIndex] = sum / 273;
    }
}
```

Ensuite, un kernel gaussien à 5 dimensions (comme suit) est utilisé pour compléter le filtrage gaussien :

```
int Gaussian[25] = { 1,4,7,4,1,
                    4,16,26,16,4,
                    7,26,41,26,7,
                    4,16,26,16,4,
                    1,4,7,4,1 }; //sum is 273
```

Nous avons effectué une opération en niveaux de gris lors de la lecture de l'image. C'est plus facile. Pour les images couleur, il faut calculer le RGB.

Pour la partie débruitage , on utilise Bilateral filter. C'est un filtre qui peut préserver les bords et le débruitage. Peut filtrer le bruit dans les données d'image, et également préserver le bord et la texture de l'image, etc.

La matrice de coefficients de gabarit du filtre bilatéral est obtenue par le coefficient de plage du produit scalaire de matrice de modèle gaussien (multiplication au niveau de l'élément). Donc on a utilisé le modèle Gaussien avant. Sur cette base, on ajoute un kernel de plage et on le multiplie pour compléter le filtrage bilatéral.

$$w(i, j, k, l) = \exp \left(-\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2} \right).$$

num est le kernel après la calculation.

```
//Prevent cross-border
if (xIndex > halfEW && xIndex < imgWidth - halfEW && yIndex > halfEH && yIndex < imgHeight - halfEH)
{
    int sum = 0;
    double num = 0;
    int sigm = 50;
    for (int i = -halfEH; i < halfEH + 1; i++)
    {
        for (int j = -halfEW; j < halfEW + 1; j++)
        {
            /*if (dataIn[(i + yIndex) * imgWidth + xIndex + j] > dataOut[yIndex * imgWidth + xIndex])
            {
                dataOut[yIndex * imgWidth + xIndex] = dataIn[(i + yIndex) * imgWidth + xIndex + j];
            }*/

            num = exp(-(double)((dataIn[(i + yIndex) * imgWidth + xIndex + j] - dataIn[yIndex * imgWidth + xIndex]) *
                (dataIn[(i + yIndex) * imgWidth + xIndex + j] - dataIn[yIndex * imgWidth + xIndex]) / sigm / sigm) / 2);
            sum += (int)dataIn[(i + yIndex) * imgWidth + xIndex + j] * d_const_Gaussian[(i + 2) * 5 + j + 2] * num;
        }
    }

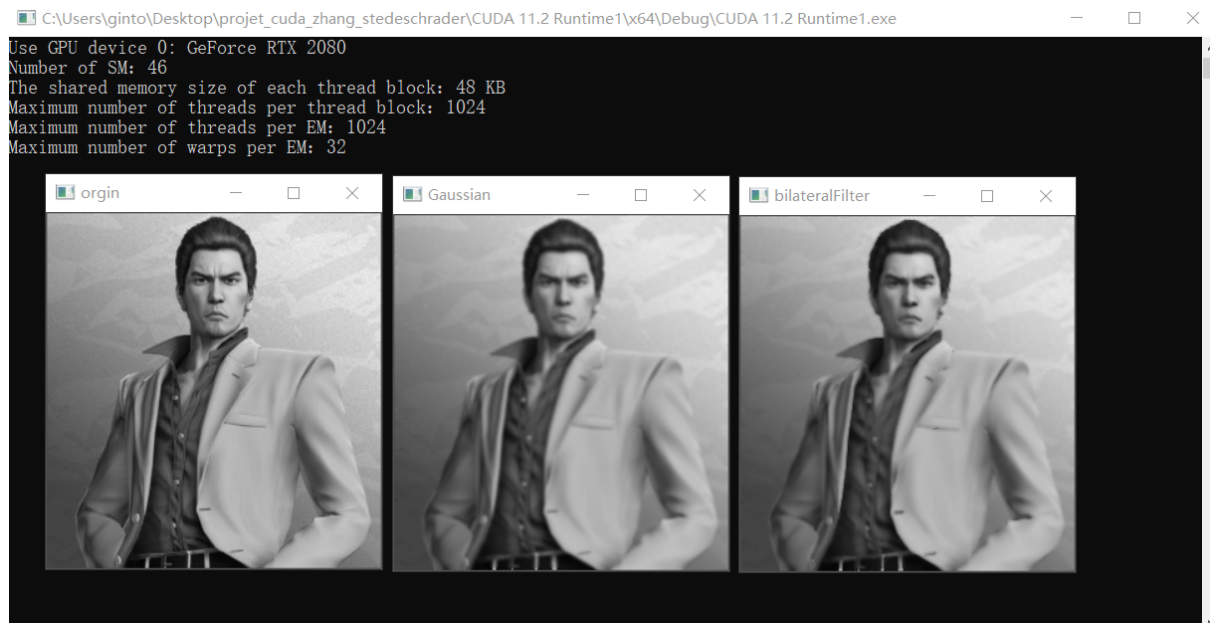
    if (sum / 273 < 0)
        dataOut[yIndex * imgWidth + xIndex] = 0;
    else if (sum / 273 > 255)
        dataOut[yIndex * imgWidth + xIndex] = 255;
    else
        dataOut[yIndex * imgWidth + xIndex] = sum / 273;
}
```

Si on met une valeur sigm différente, on pourra obtenir un résultat différent. (Ci-dessus, un exemple avec sigm=50)

Les commentaires sont bien mis dans le code.

Problèmes rencontrés : Aucun problème dans cette partie.

Résultats :



4 - Détection Sobel

Fichier : Sobel_detection_OpenCVwithCUDA.cu

Explication : On utilise Sobel operator pour faire de la détection de contours. L'opérateur comprend deux ensembles de matrices 3x3 horizontales et verticales, et les convolve avec le plan image pour obtenir une valeur approximative de la différence horizontale et verticale de luminance. Si A représente l'image d'origine, Gx et Gy représentent respectivement la valeur de gris de l'image détectée par le bord horizontal et le bord vertical. La formule est la suivante:

$$\begin{aligned} G_x &= (-1)*f(x-1, y-1) + 0*f(x,y-1) + 1*f(x+1,y-1) \\ &\quad + (-2)*f(x-1,y) + 0*f(x,y) + 2*f(x+1,y) \\ &\quad + (-1)*f(x-1,y+1) + 0*f(x,y+1) + 1*f(x+1,y+1) \\ &= [f(x+1,y-1)+2*f(x+1,y)+f(x+1,y+1)]-[f(x-1,y-1)+2*f(x-1,y)+f(x-1,y+1)] \end{aligned}$$

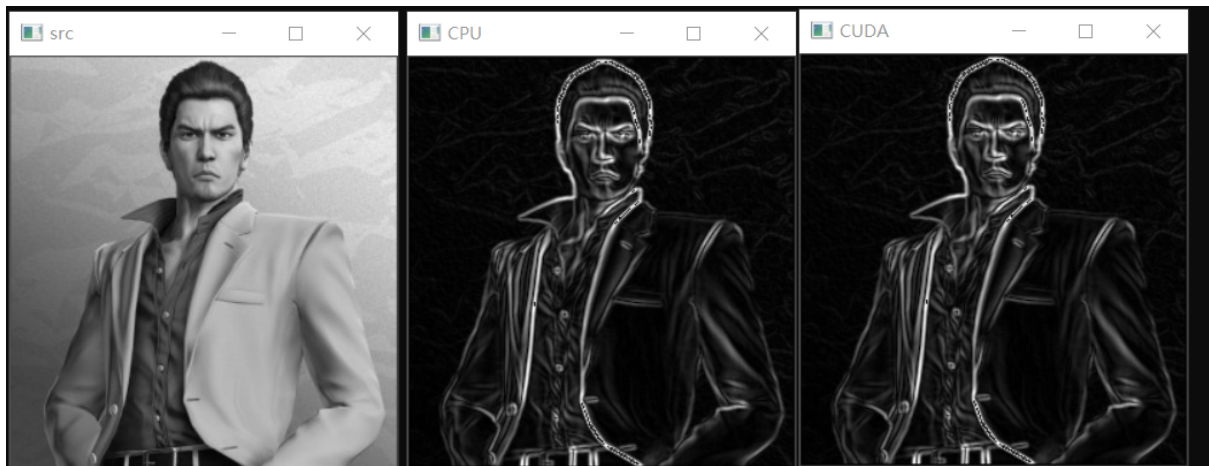
$$\begin{aligned} G_y &= 1*f(x-1, y-1) + 2*f(x,y-1) + 1*f(x+1,y-1) \\ &\quad + 0*f(x-1,y) + 0*f(x,y) + 0*f(x+1,y) \\ &\quad + (-1)*f(x-1,y+1) + (-2)*f(x,y+1) + (-1)*f(x+1, y+1) \\ &= [f(x-1,y-1) + 2*f(x,y-1) + f(x+1,y-1)]-[f(x-1, y+1) + 2*f(x,y+1)+f(x+1,y+1)] \end{aligned}$$

Puisque le principe de détection passe par la convolution de Gx et Gy dans deux directions, nous devons indexer correctement la position de chaque élément dans le petit carré 3 * 3 centré sur le pixel cible lors de la mise en œuvre de CUDA. Le transfert côté CPU vers le GPU est une mémoire continue unidimensionnelle, ce qui augmente la difficulté d'indexation. Par conséquent, dans la conception du Block et Grid, j'ai complètement mappé l'image entière sur Grid. Chaque thread correspond à un pixel. Cartographie précise de la mémoire unidimensionnelle par indexation bidimensionnelle

Les commentaires sont bien mis dans le code.

Problèmes rencontrés : Aucun problème dans cette partie

Résultats :



référence: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>

5 - Histogramme d'une image

Fichier : Histogram.cu

Explication : Pour cette partie, notre fonction Histogram_Calculation_CUDA() calcule le niveau de gris de chaque pixel de l'image (entre 0 et 255), et nous le renvoie écrit dans le terminal.

Problèmes rencontrés : Aucun problème dans cette partie

Résultats :

