BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil pengujian dapat disimpulkan :

1. Telah berhasil mengembangkan *code* program untuk segmentasi citra menggunakan metode *level set* untuk *active contour* berbasis CPU C++ dengan tingkat *error* 8,13%-15,83% dan GPU CUDA dengan tingkat *error* 3,76% - 12,62% dibandingkan dengan matlab zhang et al (2009).
2. Berdasarkan beberapa pengujian dari beberapa citra uji coba dan beberapa perangkat yang berbeda dapat disimpulkan bahwa :
   a. Menggunakan GPU NVIDIA Geforce GTX 660 dapat mempercepat proses komputasi hingga 34-42x lipat lebih cepat dibandingkan dengan proses komputasi menggunakan CPU *core* i5 3330.
   b. Menggunakan GPU NVDIA Geforce GT 635M dapat mempercepat proses komputasi hingga 17-34x lipat lebih cepat dibandingkan dengan proses komputasi menggunakan CPU *core* i5 3330.

5.2. Saran

Aplikasi dapat dikembangkan lebih lanjut sebagai bahan penelitian lebih lanjut, seperti:

a. Pengembangan menggunakan memory CUDA yang lain seperti *shared memory*, *texture memory*, yang kinerja dapat lebih cepat dan lebih efisien.
b. Pengembangan aplikasi juga dapat dikembangkan pada segmentasi citra dengan skema 3D.

DAFTAR PUSTAKA

Airouche, M., Bentabet, L. & Zelmat, M., 2009. Image Segmentation Using Active Contour Model. *Proceedings of the World Congress on Engineering*, I(5), pp.1-5.

Al-amri, S.S., Kalyankar, N.V. & D, K.S., 2010. Image Segmentation by Using Threshold Techniques. *Journal of Computing*, 2(5), pp.83-86.

Amine, K. & Farida, M.H., 2012. An Active Contour for Range Iamge Segmentation. *Internasional Journal SIPIJ*, 3(3).

Angelini, E.D., Song, T., Mensh, B.D. & Laine, A.F., 2007. Brain MRI Segmentation with Multiphase Minimal Partitioning A Comparative Study. *Hindawi Publishing Corporation International Journal of Biomedical Imaging*, 2007.

Ahn, Song Ho, Timer.cpp. http://www.songho.ca/ .2003

Ayed, I.B., Mitiche, A. & Belhadj, Z., 2005. Multiregion Level-Set Partitioning of Synthetic Aperture Radar Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5), pp.793-800.

Barman, P.C., Miah, M.S., Signh, B.C. & Khatun, T., 2011. MRI Image Segementation Using Level Set Method And Implement An Medical Diagnosis System. *Computer Science & Engineering : An Intenational Journal (CSEIJ)*, 1(5), pp.1-10.

Barney, B., 2008. *Introduction to Parallel Computing*. [Online] Available at: https://computing.llnl.gov/tutorials/parallel_comp/ [Accessed 10 April 2013].

Barney, B., 2012. *Introduction to Parallel Computing*. [Online] Available at: https://computing.llnl.gov/tutorials/parallel_comp/#Whatis [Accessed 18 April 2013].

Bhatotia, P., Rodrigo, R. & Verma, A., n.d. *Shredder : GPU Accelerated Incremental Storage and Computation*. India: Max Planck Institute for Software Systems and IBM Research.

Bhoyar, K. & Kakde, O., n.d. Color Image Segmenatation Based on JND Color Histogram. *Inaternational Journal of Image processing (IJIP)*, 3(6), pp.283-92.

Bourouis, S. & Hamrouni, K., 2011. *Deformable Model-Based Segmentation of Brain Tumor from MR Images*. Shanghai: InTech.

Brox, T. & Weickert, J., 2004. Level Set Based Image Segementation wtih Multiple Regions. *Springer LNCS 3175 in Pattern Recognition*, pp.415-23.

Caselles, V., 1995. Geometr Active Contoursic Models f. *IEEE*, pp.9-12.

Castleman, R.K., 1996. In *Digital Image Processing*. New Jersey. p.81.

Chan, T.F., Li, H., Lysaker, M. & Tai, X.-C., 2007. Level Set Method for Positron Emission Tomography. *Hindawi Publishing Corporation International Journal of Biomedical Imaging*, 2007.

Chan, T. & Zhu, W., 2005. Level Set Based Shape Prior Segmentation. *IEEE CVPR*, 2, pp.1164-70.

Chen, T.F., 2008. *Medical Image Segmentation using Level Set*. Technical Report. Cheriton School of Computer Science.

Chitade, A.Z. & Katiyar, S.K., 2010. Colour Based Image Segementation Using K-Means Clustering. *International Journal of Engineering Science and Technology*, 2(10), pp.5319-25.

Dubrovina, A. & Kimmel, R., 2012. CIS *Active Contour For Multi-Region Image Segmentastion With A Single Level Set Function*. PhD Thesis. Israel: Tehnical Report Technion.

El-Baz, A. & Gimel'farb, G., 2008. Image Segmentation with A Parametric Deformable Model Using Shape and Appearence Priors. In *IEEE Conference on INn Computer Vision and Pattern Recognition, 2008 (CVPR 2008)*. Anchorage, AK, 2008.

Gao, L., Liu, X. & Chen, W., 2012. Phase- and GVF-Based Level Set Segmentation of Ultrasonic Breast Tumors. *HIndawi Publishing Coporation Journal of Applied Mathematics*, 2012.

Ghorpade, J., Parande, J., Kulkarni, M. & Bawaskar, A., 2012. GPGPU Processing in CUDA Architecture. *Advanced Computing : An International Journal*, 3(1), pp.105-20.

Grama, A., Gupta, A., Karipys, G. & Kumar, V., 2003. In *Introduction to Parallel Computing*. Addison Wesley.

Halder, A. & Pathak, N., n.d. An Evolutionary Dynamic Clustering Based Colour Image Segmentation. *International Journal of Image Processing (IJIP)*, 4(6), pp.549-56.

Han, X., Xu, C. & Prince, J.L., 2003. A 2D Moving Grid Geometric Deformable Model. In *Proceedings 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*., 2003.

Haque, M.N. & Uddin, M.S., 2011. Accelerating Fast Fourier Transformation for Image Processing using Graphics Processing Unit. *Journal of Emerging Trends in Computing and Information Sciences*, 2(8), pp.367-75.

Iwai, K., Nishikawa, N. & Kurokawa, T., 2012. Acceleration of AES Encryption on CUDA GPU. *International Journal of Networking and Computing*, 2(1), pp.131-45.

Kass, M., Witkin, A. & Terzopoulos, D., 1987. Snakes : Active Contour Models. *Internatinal Journal of Computer Vision*, I(4), pp.321-31.

Krishnaveni, M. & Radha, V., 2011. Quantitative evaluation of Segmentation Algorithms Based on Level Set Method for ISL Datasets. *International Journal - IJCSE*, 3(4), pp.2361-69.

Kumar, P. et al., 2012. Improving Medical Image Segmentation Tecniques Using Multiphase Using Multiphase level Set Approach Via Bias Correction. *International Journal of Engineering and Advanced Technology (IJEAT)*, 1(5), pp.285-89.

Lazrag, H. & Naceur, M.S., 2012. Combination of the Level-Set Methods with the Contourlet Transform for the Segmentation of the IVUS Images. *Hindawi Puclising Corporation International Journal of Biomedical Imaging* , 2012.

Lee, C.P., 2005. *Robust Image Segmentation using Active Contours : Level Set Approaches*. PhD Thesis. North Carolina University.

Li, C. et al., 2011. A Level Set Method for Image Segmentation in the Presence of Intentsity Inhomogenities with Application to MRI. *IEEE : Transactions On Image Processing*, 20(7), pp.2007-16.

Li, C., Kao, C., Gore, J.C. & Ding, Z., 2007. Implicit Active Contours Driven by Local Binary Fitting Energy. *IEEE Conference on Computer Vision and Pattern Recognition*, 17, pp.1940-49.

Lin, C.-H., Liu, C.-H., Chang, S.-C. & Hon, W.-K., 2012. Memory-Efficient Pattern Matching Architectures using Perfect Hashing on Graphic Processing Units. In *2012 Proceedings IEEE INFOCOM*. Orlando, FL, 2012.

Li, C., Xu, C., Gui, C. & Fox, M.F., 2005. Level Set Evolution Wtihout Re-initialization : A New Variational Formulation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5.

Li, C., Xu, C., Gui, C. & Fox, M.D., 2010. Distance Regularized Level Set Evolution and Its Application to Image Segmentation. *IEEE Transactions on Image Processing*, 19(12), pp.3243-54.

Mostofi, H., 2009. *Fast Level Set Segmentation of Biomedical Images using Graphics*. London: University of Oxford : Department of Engineering Science.

Munim, E. & Abd, H.E., 2007. Curve/Surface Representation and Evolution using Vector Level Sets with Application to The Shape Based Segmentation Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), pp.945-58.

Murinto & Harjoko, A., 2009. Segemntasi Citra Menggunakan Watershed dan Intensitas Filtering Sebagai Pre Processing. *Seminar Nasional Informatika 2009 (semnasIF2009)*, pp.A-43 - A-47.

Ni, K., Bresson, X., Chan, T. & Esedoglu, S., 2009. Local Histogram Based Segmentation Using the Wasserstein Distance. *Int J Compt Vis*, pp.97-111.

NVIDIA, 2013. *What is GPU Computing?* [Online] Available at: http://www.nvidia.com/object/what-is-gpu-computing.html [Accessed 10 April 2013].

Osher, S. & Fedkiw, R., 2002. In *Level Set Method and Dynamic Implicit Surfaces*.

Owens, J., Davis, U., Luebke, D. & NVIDIA, 2013. *What is CUDA?* [Online] Available at: http://www.nvidia.com/object/cuda_home_new.html [Accessed 10 April 2013].

Peng, D. et al., 1999. A PDE Based Fast Local Level Set Method. *Journal of Computational Physics* , pp.410-38.

Raimondo, F., Kamienkowski, J.E., Sigman, M. & Slezak, D.F., 2012. CUDAICA: GPU Optimization of Infomax-ICA EEG Analysis. *Hindawi Publishing Corporation Computational Intelligence and Neuroscience*, 2012.

Rocca, M.R.D., Fiani, M., Fortunato, A. & Pistillo, P., n.d. *Active Contour Model to Detect Linear Features in Satellite Images*. Fisciano: Dipartimento di Ingegneria Civile.

Roodt, Y., Visser, W. & Clarke, W.A., 2007. Image Processing on the GPU: Implementing the Canny Edge. *GPU Computing : Computer Vision*.

Senthilkumaran, N. & Rajesh, R., 2009. Edge Detection Techniques for Image Segmentation - A Survey of Soft Computing Approaches. *International Journal of Recent Trends in Engineering*, 1(2), pp.250-54.

Soenarwo, H., 2004. *C/C++ Programming*. IndoProg.

Varray, F., Cachard, C., Ramali, A. & Basset, O., 2011. Simulation of Ultrasound Nonlinear Propagation on GPU using a Generalized Angular Spectrum Method. *EURASIP Journal on Image and Video Processing*, 17.

Vashist, P. & Hema, K., 2013. Watershed Transform on Image Segmentation and Data Classification. *International Journal of Science and Tehcnology (IJST)*, 2(1), pp.112-21.

Wang, Y., Wei, G.-W. & Yang, S., 2012. Selective Extraction of Entangled Textures via Adaptive PDE Transform. *Hindawi Publishing Corporation International Journal of Biomedical Imaging*, 2012.

Yu, W. et al., 2008. Level Set Segmentation of Cellular Images Based on Topological Dependence. *ISVC*, 1, pp.540-51.

Zaher, S., Badr, A., Farang, I. & AbdElmaged, T., 2012. Using P System with GPU Model to Design and Implement a Public Key Cryptography. *International Journal of Computer Applications*, 60(6).

Zhang, K., Song, H. & Zhang, L., 2009. Active Contours Driven by Local Image Fitting Energy. *ELSEVIER*, 43(8), pp.1199-206.

LAMPIRAN

LAMPIRAN A

PROGRAM *LEVEL SET LIF* C PROGRAMMING

A1. main.cpp

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <float.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include "Timer.h"
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

#define IMAGE   "b_256.bmp"
#define KEY_ESCAPE 27

const int iterasi    = 5000;
const int batas          = 500;
const int status     = 1;
const char      *type           = ".txt";
const char      *nama_txt_time = "time_b_256.txt";
const float     epsilon    = 1.5;
const float     timestep   = 1;
const float     sigma      = 3;
const float     sigma_phi  = 0.5;
const char*image_path      = IMAGE;
const int kernel_size      = 5;
const int kernel_size_phi = 5;

#ifdef _WIN32
#   pragma warning( disable : 4996 ) // disable deprecated warning
#endif

#pragma pack(1)

typedef struct{
    short type;
    int size;
    short reserved1;
    short reserved2;
    int offset;
} BMPHeader;
```

```
typedef struct{
    int size;
    int width;
    int height;
    short planes;
    short bitsPerPixel;
    unsigned compression;
    unsigned imageSize;
    int xPelsPerMeter;
    int yPelsPerMeter;
    int clrUsed;
    int clrImportant;
} BMPInfoHeader;

//Isolated definition
typedef struct{
    unsigned char x, y, z, w;
} uchar4;

typedef struct {
    int width;
     int height;
     char* title;

     float field_of_view_angle;
     float z_near;
     float z_far;
} glutWindow;

glutWindow win;

//variabel global
float *D_in;
float *D_out;
float *phi;
float *phi_out;
float *heaveside_phi;
float *h_phi;
float *diract_phi;
float *m1;
float *m2;
float *f1;
float *f2;
float *c1;
float *c2;
float *f1_out;
float *f2_out;
float *c1_out;
float *c2_out;
float *gaussian_matrix;
float *gaussian_matrix_phi;
float *coba_konv;

uchar4    *h_Src;
```

```
int   imageW;
int   imageH;
int   N;
int         row;
int   col;
int   its=0;
floattime1;
floattime2;
floatsum = 0;

//fungsi-fungsi
void init_phi();
void LoadBMPFile(uchar4 **dst, int *width, int *height, const char
*name);
void kernel_gaussian(float *kernel, float sigma, int kernel_size);
void konvolusi_CPU(float *pixel_data_in,float *pixel_data_out,
     float *kernel, int kernel_size, float sigma);
void delta_h(float *phi, float *diract_phi, float epsilon);
void heaviside(float *phi, float *h_phi, float epsilon);
void update_c(float *h_phi, float *c, int status);
void update_f(float *D, float *c, float *f);
void update_m(float *f, float *c, float *m);
void update_phi();
void save_to_file(float *data_array, const char *name);
void free();
void display(void);
void init();


int main(int argc, char** argv)
{
     FILE *fp;
     Timer t;
     int c = 0;

     //memanggil fungsi LoadBMPfile
     LoadBMPFile(&h_Src, &imageW, &imageH, image_path);

     //daftar memory untuk array yang digunakan
     if((D_in = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
     {
          printf("\nD_in NULL\n");
     }
     if((D_out = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
     {
          printf("\nD_out NULL\n");
     }
     if((phi = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
     {
          printf("\nphi NULL\n");
     }
     if((phi_out = (float *)malloc(imageW*imageH*sizeof(float)))
          ==NULL)
     {
```

```
      printf("\nphi_out NULL\n");
}
if((m1 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nm1 NULL\n");
}
if((m2 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nm2 NULL\n");
}
if((f1 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nf1 NULL\n");
}
if((f2 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nf2 NULL\n");
}
if((c1 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nc1 NULL\n");
}
if((c2 = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nc2 NULL\n");}
if((f1_out = (float *)malloc(imageW*imageH*sizeof(float)))
      ==NULL)
{
      printf("\nf1_out NULL\n");
}
if((f2_out = (float *)malloc(imageW*imageH*sizeof(float)))
      ==NULL)
{
      printf("\nf2_out NULL\n");}
if((c1_out = (float *)malloc(imageW*imageH*sizeof(float)))
      ==NULL)
{
      printf("\nc1_out NULL\n");
}
if((c2_out = (float *)malloc(imageW*imageH*sizeof(float)))
      ==NULL)
{
      printf("\nc2_out NULL\n");
}
if((coba_konv = (float *)malloc(imageW*imageH*sizeof(float)))
      ==NULL)
{
      printf("\nc2 NULL\n");}
 if((heaveside_phi =
            (float*)malloc(imageW*imageH*sizeof(float)))==NULL)
{
      printf("\nheaveside NULL\n");
}
if((h_phi = (float *)malloc(imageW*imageH*sizeof(float)))==NULL)
```

```
{
     printf("\nh_phi NULL\n");
}
if((diract_phi = (float *)malloc(imageW*imageH*sizeof(float)))
     ==NULL)
{
     printf("\ndiract_phi\n");
}
if((gaussian_matrix=(float *)malloc
     (kernel_size*kernel_size*sizeof(float)))==NULL)
{
     printf("\gaussian_matrix\n");
}
if((gaussian_matrix_phi=(float *)malloc
     (kernel_size_phi*kernel_size_phi*sizeof(float)))==NULL)
{
     printf("\gaussian_matrix_phi\n");
}

//membaca nilai pixel
for(row=0;row<imageW;row++)
{
     for(col=0;col<imageH;col++)
     {
          D_in[row*imageW+col] = h_Src[row*imageW+col].x;
     }
}

//inisialisasi nilai phi
init_phi();

//inisialisasi kernel gaussian untuk konvolusi pada fungsi
     rectangular window
kernel_gaussian(gaussian_matrix, sigma, kernel_size);

//inisialisasi kernel gaussian untuk konvolusi pada
     nilai phi baru
kernel_gaussian(gaussian_matrix_phi, sigma_phi,
     kernel_size_phi);

//memulai perhitungan waktu
t.start();

//update phi dengan jumlah iterasi yang ditentukan pada
        varibel iterasi
for(its=0;its<iterasi;its++)
{
     //memanggil prosedur update_phi()
     update_phi();

     c += 1;
     if(c % batas == 0)
     {
          // menyimpan data waktu pada batas yang ditentukan
```

```
                if(status == 1)
                {
                      time2 = t.getElapsedTimeInMilliSec();
                      if((fp=fopen(nama_txt_time,"a")) == NULL)
                      {
                            printf("Cannot open file.\n");
                            exit(1);
                      }
                      fprintf (fp, "\n");
                      fprintf (fp, "Time iterasi %d = %0.2f detik", c,
                      (time2/1000));
                      fprintf (fp, "\n");
                }
                //menyimpan data phi pada file txt pada batas yang
                ditentukan
                else if (status == 2)
                {
                      printf("\niterasi ke %d : %f ms\n", c, time2);
                      sprintf(filename, "%s%d%s", one,c,type);
                      save_to_file(phi,filename);
                }
          }
          printf("\nITERASI = %d ... SELESAI\n",its);

          //waktu perhitungan berhenti
          t.stop();
          time1 = t.getElapsedTimeInMilliSec();
          printf("\nNormal LIF : %f ms\n", time1);

          //mengosongkan memory dengan memanggil prosedur free();
          free();

          getch();
}


void LoadBMPFile(uchar4 **dst, int *width, int *height, const char
*name)
{

          /*
           * Copyright 1993-2007 NVIDIA Corporation.  All rights reserved.
           *
           * NOTICE TO USER:
           *
           * This source code is subject to NVIDIA ownership rights under
          U.S. and
           * international Copyright laws.  Users and possessors of this
          source code
           * are hereby granted a nonexclusive, royalty-free license to use
          this code
           * in individual and commercial software.
           *
```

```
 * NVIDIA MAKES NO REPRESENTATION ABOUT THE SUITABILITY OF THIS
SOURCE
 * CODE FOR ANY PURPOSE.  IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR
 * IMPLIED WARRANTY OF ANY KIND.  NVIDIA DISCLAIMS ALL WARRANTIES
WITH
      * REGARD TO THIS SOURCE CODE, INCLUDING ALL IMPLIED
WARRANTIES OF
 * MERCHANTABILITY, NONINFRINGEMENT, AND FITNESS FOR A PARTICULAR
PURPOSE.
 * IN NO EVENT SHALL NVIDIA BE LIABLE FOR ANY SPECIAL, INDIRECT,
INCIDENTAL,
 * OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING
FROM LOSS
 * OF USE, DATA OR PROFITS,  WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE
 * OR OTHER TORTIOUS ACTION,  ARISING OUT OF OR IN CONNECTION WITH
THE USE
 * OR PERFORMANCE OF THIS SOURCE CODE.
 *
 * U.S. Government End Users.   This source code is a "commercial
item" as
 * that term is defined at  48 C.F.R. 2.101 (OCT 1995), consisting
of
 * "commercial computer  software"  and "commercial computer
software
 * documentation" as such terms are  used in 48 C.F.R. 12.212 (SEPT
1995)
 * and is provided to the U.S. Government only as a commercial end
item.
 * Consistent with 48 C.F.R.12.212 and 48 C.F.R. 227.7202-1 through
 * 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire
the
 * source code with only those rights set forth herein.
 *
 * Any use of this source code in individual and commercial
software must
 * include, in the user documentation and internal comments to the
code,
 * the above Disclaimer and U.S. Government End Users Notice.
 */

BMPHeader hdr;
BMPInfoHeader infoHdr;
int x, y;

FILE *fd;


printf("Loading %s...\n", name);
if(sizeof(uchar4) != 4){
    printf("***Bad uchar4 size***\n");
    exit(0);
}
```

```
    if( !(fd = fopen(name,"rb")) ){
        printf("***BMP load error: file access denied***\n");
        exit(0);
    }

    fread(&hdr, sizeof(hdr), 1, fd);
    if(hdr.type != 0x4D42){
        printf("***BMP load error: bad file format***\n");
        exit(0);
    }
    fread(&infoHdr, sizeof(infoHdr), 1, fd);

    if(infoHdr.bitsPerPixel != 24){
        printf("***BMP load error: invalid color depth***\n");
        exit(0);
    }

    if(infoHdr.compression){
        printf("***BMP load error: compressed image***\n");
        exit(0);
    }

    *width  = infoHdr.width;
    *height = infoHdr.height;
    *dst    = (uchar4 *)malloc(*width * *height * 4);

    printf("BMP width: %u\n", infoHdr.width);
    printf("BMP height: %u\n", infoHdr.height);

    fseek(fd, hdr.offset - sizeof(hdr) - sizeof(infoHdr), SEEK_CUR);

    for(y = 0; y < infoHdr.height; y++){
        for(x = 0; x < infoHdr.width; x++){
            (*dst)[(y * infoHdr.width + x)].z = fgetc(fd);
            (*dst)[(y * infoHdr.width + x)].y = fgetc(fd);
            (*dst)[(y * infoHdr.width + x)].x = fgetc(fd);
        }

        for(x = 0; x < (4 - (3 * infoHdr.width) % 4) % 4; x++)
            fgetc(fd);
    }
    if(ferror(fd)){
        printf("***Unknown BMP load error.***\n");
        free(*dst);
        exit(0);
    }else
        printf("BMP file loaded successfully!\n");

    fclose(fd);
}

void kernel_gaussian(float *kernel, float sigma, int kernel_size)
{
    float r, s = 2.0 * sigma * sigma;
```

```
    float sum = 0.0;
    int half = (kernel_size-1)/2;

    for(int x = (-1*half); x <= half; x++)
    {
        for(int y = (-1*half); y <= half; y++)
        {
            r = sqrt((float)(x*x + y*y));
            kernel[kernel_size*(x+half)+(y+half)] =
                                (exp(-(r*r)/s))/(M_PI * s);
            sum += kernel[kernel_size*(x+half)+(y+half)];
      }
    }

    for(int i = 0; i < kernel_size; ++i)
    {
        for(int j = 0; j < kernel_size; ++j)
        {
           kernel[kernel_size*i+j] /= sum;
        }
     }
}

void konvolusi_CPU(float *pixel_data_in,float *pixel_data_out,
     float *kernel, int kernel_size, float sigma)
{
    float sum = 0.0;
     int half = (kernel_size-1)/2;
     int row_i=0;
     int col_j=0;
     float temp = 0.0;

     for(row=0;row<imageW;row++)
     {
          for(col=0;col<imageH;col++)
          {
               sum = 0;
               for(int i=-half;i<=half;i++)
               {
                    row_i = row + i;
                    for(int j=-half;j<=half;j++)
                    {
                         col_j = col + j;
                         temp =0;
                         if((col_j>=0)&&(col_j<imageH)
                              &&(row_i>=0)&&(row_i<imageW))
                         {
                              temp=pixel_data_in[row_i*imageW+col_j];
                         }
                         sum+=temp*
                              kernel[(i+half)*kernel_size+(j+half)];
                    }
               }
               pixel_data_out[row*imageW+col] = sum;
```

```
            }
        }
}

void heaviside(float *phi, float *h_phi, float epsilon)
{
        for(int row=0;row<imageW;row++)
        {
            for(int col=0;col<imageH;col++)
            {
                h_phi[row*imageW+col] =
               0.5 * (1 + (2/M_PI) * atan(phi[row*imageW+col]/epsilon));
            }
        }
}

void update_c(float *h_phi, float *c, int status)
{
        for(int row=0;row<imageW;row++)
        {
            for(int col=0;col<imageH;col++)
            {
                if(status == 1)
                {
                    c[row*imageW+col] = h_phi[row*imageW+col];
                }
                else
                {
                    c[row*imageW+col] = 1-h_phi[row*imageW+col];
                }
            }
        }
}

void update_f(float *D, float *c, float *f)
{
        for(int row=0;row<imageW;row++)
        {
            for(int col=0;col<imageH;col++)
            {

                f[row*imageW+col] =
                    D[row*imageW+col]*c[row*imageW+col];

            }
        }
}

void update_m(float *f, float *c, float *m)
{
        for(int row=0;row<imageW;row++)
        {
            for(int col=0;col<imageH;col++)
            {
```

```
                m[row*imageW+col] =
                    f[row*imageW+col]/c[row*imageW+col];
            }
        }
}
void update_phi()
{
        delta_h(phi,diract_phi,epsilon);
        heaviside(phi,heaveside_phi,epsilon);
        update_c(heaveside_phi,c1,1);
        update_c(heaveside_phi,c2,2);
        update_f(D_in,c1,f1);
        update_f(D_in,c2,f2);
        konvolusi_CPU(c1,c1_out,gaussian_matrix,kernel_size,sigma);
        konvolusi_CPU(c2,c2_out,gaussian_matrix,kernel_size,sigma);
        konvolusi_CPU(f1,f1_out,gaussian_matrix,kernel_size,sigma);
        konvolusi_CPU(f2,f2_out,gaussian_matrix,kernel_size,sigma);
        update_m(f1_out,c1_out,m1);
        update_m(f2_out,c2_out,m2);
        for(row=0;row<imageW;row++)
        {
            for(col=0;col<imageH;col++)
            {
                N = row*imageW+col;
                phi_out[N] = phi[N] + timestep * diract_phi[N] *
                            ((D_in[N]-(m1[N]*heaveside_phi[N])-(m2[N]*
                            (1-heaveside_phi[N])))*(m1[N]-m2[N]));
            }
        }
        konvolusi_CPU(phi_out, phi_out, gaussian_matrix_phi,
                                        kernel_size_phi, sigma_phi);
        phi = phi_out;
}
void delta_h(float *phi, float *diract_phi, float epsilon)

{
        for(int row=0;row<imageW;row++)
        {
            for(int col=0;col<imageH;col++)
            {
                diract_phi[row*imageW+col] =
                    epsilon/(M_PI*((epsilon*epsilon)+
                    (phi[row*imageW+col]*phi[row*imageW+col])));
            }
        }
}
void save_to_file(float *data_array, const char *name)
{
        FILE *fp;
        if((fp=fopen(name,"w")) == NULL)
        {
            printf("Cannot open file.\n");
            exit(1);
        }
```

```
	for(row=0;row<imageW;row++)
	{
		for(col=0;col<imageH;col++)
		{
			/* write to file */
			fprintf(fp,"%4.4f",
				data_array[((imageW-1)-row)*imageW+col]);
			fprintf(fp,"\t");
		}
		fprintf(fp,"\n");
	}
	printf("\n Write file..  %s  SUKSES.....\n", name);
}
void free()
{
	free(D_in);
	free(D_out);
	free(phi);
	free(phi_out);
	free(m1);
	free(m2);
	free(f1);
	free(f2);
	free(c1);
	free(c2);
	free(f1_out);
	free(f2_out);
	free(c1_out);
	free(c2_out);
	free(heaveside_phi);
	free(h_phi);
	free(diract_phi);
	free(gaussian_matrix);
	free(gaussian_matrix_phi);
}
void init_phi()
{
	printf("\nInisialisasi phi... SUKSES...\n");
	for(row=0;row<imageW;row++)
	{
		for(col=0;col<imageH;col++)
		{
			if((row<15)||(col<15)||(row>(imageW-15))||
				(col>(imageH-15)))
			{
				phi[row*imageW+col] = 1;
			}
			else
			{
				phi[row*imageW+col] = -1;
			}
		}
	}
}
```

A2. time.cpp

```
//////////////////////////////////////////////////////////////////////
//////
// Timer.cxx
// =========
// High Resolution Timer.
// This timer is able to measure the elapsed time with 1 micro-second
accuracy
// in both Windows, Linux and Unix system
//
//  AUTHOR: Song Ho Ahn (song.ahn@gmail.com)
// CREATED: 2003-01-13
// UPDATED: 2006-01-13
//
// Copyright (c) 2003 Song Ho Ahn
//////////////////////////////////////////////////////////////////////
//////

#include "Timer.h"
#include <stdlib.h>

//////////////////////////////////////////////////////////////////////
///////
// constructor
//////////////////////////////////////////////////////////////////////
///////
Timer::Timer()
{
#ifdef WIN32
    QueryPerformanceFrequency(&frequency);
    startCount.QuadPart = 0;
    endCount.QuadPart = 0;
#else
    startCount.tv_sec = startCount.tv_usec = 0;
    endCount.tv_sec = endCount.tv_usec = 0;
#endif

    stopped = 0;
    startTimeInMicroSec = 0;
    endTimeInMicroSec = 0;
}



//////////////////////////////////////////////////////////////////////
///////
// distructor
//////////////////////////////////////////////////////////////////////
///////
Timer::~Timer()
{
}
```

```
////////////////////////////////////////////////////////////////////////
///////
// start timer.
// startCount will be set at this point.
////////////////////////////////////////////////////////////////////////
///////
void Timer::start()
{
    stopped = 0; // reset stop flag
#ifdef WIN32
    QueryPerformanceCounter(&startCount);
#else
    gettimeofday(&startCount, NULL);
#endif
}


////////////////////////////////////////////////////////////////////////
///////
// stop the timer.
// endCount will be set at this point.
////////////////////////////////////////////////////////////////////////
///////
void Timer::stop()
{
    stopped = 1; // set timer stopped flag

#ifdef WIN32
    QueryPerformanceCounter(&endCount);
#else
    gettimeofday(&endCount, NULL);
#endif
}


////////////////////////////////////////////////////////////////////////
///////
// compute elapsed time in micro-second resolution.
// other getElapsedTime will call this first, then convert to correspond
resolution.
////////////////////////////////////////////////////////////////////////
///////
double Timer::getElapsedTimeInMicroSec()
{
#ifdef WIN32
    if(!stopped)
        QueryPerformanceCounter(&endCount);

    startTimeInMicroSec = startCount.QuadPart * (1000000.0 /
frequency.QuadPart);
    endTimeInMicroSec = endCount.QuadPart * (1000000.0 /
frequency.QuadPart);
```

```
#else
     if(!stopped)
          gettimeofday(&endCount, NULL);

     startTimeInMicroSec = (startCount.tv_sec *1000000.0) +
          startCount.tv_usec;
     endTimeInMicroSec = (endCount.tv_sec * 1000000.0) +
          endCount.tv_usec;
#endif

    return endTimeInMicroSec - startTimeInMicroSec;
}



////////////////////////////////////////////////////////////////////////
///////
// divide elapsedTimeInMicroSec by 1000
////////////////////////////////////////////////////////////////////////
///////
double Timer::getElapsedTimeInMilliSec()
{
    return this->getElapsedTimeInMicroSec() * 0.001;
}



////////////////////////////////////////////////////////////////////////
///////
// divide elapsedTimeInMicroSec by 1000000
////////////////////////////////////////////////////////////////////////
///////
double Timer::getElapsedTimeInSec()
{
    return this->getElapsedTimeInMicroSec() * 0.000001;
}



////////////////////////////////////////////////////////////////////////
///////
// same as getElapsedTimeInSec()
////////////////////////////////////////////////////////////////////////
///////
double Timer::getElapsedTime()
{
    return this->getElapsedTimeInSec();
}
```

LAMPIRAN B

PROGRAM LEVEL SET LIF CUDA PROGRAMMING

main.cu

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <float.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include <iostream>
#include <fstream>
#include <iomanip>
#include "time.h"
#include "cuda.h"
#include "cuda_runtime.h"
#include "../common/book.h"
#include "../common/cpu_anim.h"

using namespace std;

#define IMAGE "c_512.bmp" //patch image

const int iterasi    = 40000;
const int batas      = 2000;
const int status     = 1; //1 waktu, 2 phi
const char* one      = "phi_c_512_";
const char* type     = ".txt";
const char* nama_txt_phi_awal  = "phi_c_512_0.txt";
const char* nama_txt_time      = "time_c_512.txt";
const int x = 16;
const int y = 16;
char filename[100];

//variabel global
float *D_in, *dev_D_in;
float *D_out, *dev_D_out;
float *phi, *phi_in, *phi_out,*dev_phi_in, *dev_phi_out;
float *h_phi_in, *dev_h_phi_in;
float *h_phi_out, *dev_h_phi_out;
float *diract_phi, *dev_diract_phi;
float *m1, *dev_m1;
float *m2, *dev_m2;
float *f1, *dev_f1, *dev_f1_out;
float *f2, *dev_f2, *dev_f2_out;
float *c1, *c1_out, *dev_c1, *dev_c1_out;
float *c2, *c2_out, *dev_c2, *dev_c2_out;
float *coba_konv, *dev_coba_konv;
float *gaussian_matrix, *dev_gaussian_matrix;
```

```
float *gaussian_matrix_phi, *dev_gaussian_matrix_phi;

uchar4     *h_Src;
int        imageSize;
int        N;
int        row;
int        col;
int        imageW;
int        imageH;
int        DIM;
int        its=0;
float      time1;
float      sum = 0;

const float     epsilon          = 1.5;
const float     timestep         = 1;
const float     sigma            = 3;
const float     sigma_phi        = 0.5;
const char      *image_path      = IMAGE;
const int       kernel_size      = 5;
const int       kernel_size_phi  = 5;

void LoadBMPFile(uchar4 **dst, int *width, int *height,
     const char *name);
void init_phi(float *phi);
void kernel_gaussian(float *kernel, float sigma, int kernel_size,
     const char *name);
void konvolusi_CPU(float *pixel_data_in,float *pixel_data_out,
     float *kernel, int kernel_size, float sigma);
void pixel_data_h(float *pixel_data_in, float *pixel_data_out,
     float *h_phi_out, int status);
void heaviside(float *phi, float *h_phi_out, float epsilon);
void m(float *pixel_data_in, float *pixel_data_out, float *h_phi_out,
     float *m, int status);
void delta_h(float *phi, float *diract_phi, float epsilon);
void save_to_file(float *data_array, const char *name);
void update_data_phi_CPU();
void free_CPU();
void free_GPU();
void save_result_to_txt();
void display(void);
void init();

__global__ void update_phi_GPU(float *phi_in,float *phi_out,
     float *diract_phi, float *h_phi_in, float *m1, float *m2,
     float *D_in, float timestep, int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int N = x*DIM+y;

     phi_out[N] = phi_in[N] + timestep * diract_phi[N] *
     ((D_in[N]-(m1[N]*h_phi_in[N])-(m2[N]*(1-h_phi_in[N])))*
```

```
     (m1[N]-m2[N]));
}
__global__ void delta_h_GPU(float *phi_in, float *diract_phi,
     float epsilon, int DIM, float PI)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int N = x*DIM+y;

     diract_phi[N] = epsilon/(PI*((epsilon*epsilon)+
          (phi_in[N]*phi_in[N])));
}

__global__ void heaviside_GPU(float *phi_in, float *h_phi_in,
     float *h_phi_out,float epsilon, int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;
     int N = x*DIM+y;
     h_phi_in[N] = 0.5 * (1 + (2/M_PI) * atan(phi_in[N]/epsilon));
     h_phi_out[N] = h_phi_in[N];
}

__global__ void konvolusi_GPU(float *pixel_data_in,
     float *pixel_data_out, float *kernel_gaussian, int kernel_size,
     int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int x_i = 0;
     int y_j = 0;
     float sum = 0.0;
     float temp = 0.0;
     int half = (kernel_size-1)/2;

     for (int i=(-1*half);i<=half;++i)
     {
          x_i = x+i;
          for(int j=(-1*half);j<=half;++j)
          {
               y_j = y+j;
               temp = 0;
               if((x_i>=0)&&(x_i<DIM)&&(y_j>=0)&&(y_j<DIM))
               {
                    int N = y_j*DIM+x_i;
                    temp = pixel_data_in[N];
               }
               sum += (temp *
                    kernel_gaussian[(i+half)*kernel_size+(j+half)]);
          }
     }
     pixel_data_out[y*DIM+x] = sum;
```

```
}

__global__ void update_c_GPU(float *h_phi, float *c, int status,
     int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int N = x*DIM+y;

     if(status==1)
     {
          c[N] = h_phi[N];
     }
     else
     {
          c[N] = 1 - h_phi[N];
     }
}

__global__ void update_f_GPU(float *D, float *c, float *f, int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int N = x*DIM+y;

     f[N] = D[N]*c[N];
}

__global__ void update_m_GPU(float *f, float *c, float *m, int DIM)
{
     int x = threadIdx.x + blockIdx.x * blockDim.x;
     int y = threadIdx.y + blockIdx.y * blockDim.y;

     int N = x*DIM+y;

     m[N] = f[N]/c[N];
}

void update_data_phi_GPU()
{
     DIM = imageW;
     dim3 thread (x,y) ;
     dim3 block (imageW /x,imageH /y) ;

     delta_h_GPU<<<block,thread>>>(dev_phi_in, dev_diract_phi,epsilon,
          DIM, M_PI);
     heaviside_GPU<<<block,thread>>>(dev_phi_in,dev_h_phi_in,
          dev_h_phi_out,epsilon,DIM);
     update_c_GPU<<<block,thread>>>(dev_h_phi_in, dev_c1, 1, DIM);
     update_c_GPU<<<block,thread>>>(dev_h_phi_in, dev_c2, 2, DIM);
     update_f_GPU<<<block,thread>>>(dev_D_in, dev_c1, dev_f1, DIM);
     update_f_GPU<<<block,thread>>>(dev_D_in, dev_c2, dev_f2, DIM);
```

```
        konvolusi_GPU<<<block,thread>>>(dev_c1,dev_c1_out,
             dev_gaussian_matrix,kernel_size,DIM);
        konvolusi_GPU<<<block,thread>>>(dev_c2,dev_c2_out,
             dev_gaussian_matrix,kernel_size,DIM);
        konvolusi_GPU<<<block,thread>>>(dev_f1,dev_f1_out,
             dev_gaussian_matrix,kernel_size,DIM);
        konvolusi_GPU<<<block,thread>>>(dev_f2,dev_f2_out,
             dev_gaussian_matrix,kernel_size,DIM);
        update_m_GPU<<<block,thread>>>(dev_f1_out, dev_c1_out,
             dev_m1, DIM);
        update_m_GPU<<<block,thread>>>(dev_f2_out, dev_c2_out,
             dev_m2, DIM);
        update_phi_GPU<<<block, thread>>> (dev_phi_in, dev_phi_out,
             dev_diract_phi, dev_h_phi_in, dev_m1, dev_m2,dev_D_in,
             timestep, DIM);
        konvolusi_GPU<<<block,thread>>>(dev_phi_out,dev_phi_out,
             dev_gaussian_matrix_phi,kernel_size_phi,DIM);
        dev_phi_in = dev_phi_out;
    }


    int main( void )
    {
        LoadBMPFile(&h_Src, &imageW, &imageH, image_path);
        imageSize = imageW * imageH * sizeof(float);
        DIM = imageW;
        float   elapsedTime;
        FILE *fp;
        int c = 0;

        cudaEvent_t     start, stop;
        HANDLE_ERROR( cudaEventCreate( &start ) );
        HANDLE_ERROR( cudaEventCreate( &stop ) );

        if((D_in=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
        {
             printf("\nD_in NULL\n");
        }
        if((D_out=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
        {
             printf("\nD_out NULL\n");
        }
        if((phi=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
        {
             printf("\nphi NULL\n");
        }
        if((phi_in=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
        {
             printf("\nphi_in NULL\n");
        }
        if((phi_out=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
        {
             printf("\nphi_out NULL\n");
        }
```

```
if((m1=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nm1 NULL\n");
}
if((m2=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nm2 NULL\n");
}
if((f1=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nf1 NULL\n");
}
if((f2=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nf2 NULL\n");
}
if((c1=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nc1 NULL\n");
}
if((c2=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nc2 NULL\n");
}
if((c1_out=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nc1 NULL\n");
}
if((c2_out=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nc2 NULL\n");
}
if((h_phi_in=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nh_phi_in NULL\n");
}
if((h_phi_out=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nh_phi_out NULL\n");
}
if((diract_phi=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\ndiract_phi\n");
}
if((gaussian_matrix=(float *)malloc
     (kernel_size*kernel_size*sizeof(float)))==NULL)
{
     printf("\gaussian_matrix\n");
}
if((gaussian_matrix_phi=(float *)malloc
     (kernel_size_phi*kernel_size_phi*sizeof(float)))==NULL)
{
     printf("\gaussian_matrix_phi\n");
}
```

```
if((coba_konv=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
{
     printf("\nc2 NULL\n");
}

HANDLE_ERROR(cudaMalloc((void **)&dev_D_in, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_D_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_phi_in, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_phi_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_h_phi_in, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_h_phi_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_diract_phi, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_m1, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_m2, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_f1, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_f2, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_c1, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_c2, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_f1_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_f2_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_c1_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_c2_out, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_coba_konv, imageSize));
HANDLE_ERROR(cudaMalloc((void **)&dev_gaussian_matrix,
     kernel_size*kernel_size*sizeof(float)));
HANDLE_ERROR(cudaMalloc((void **)&dev_gaussian_matrix_phi,
     kernel_size_phi*kernel_size_phi*sizeof(float)));

for(row=0;row<imageW;row++)
{
     for(col=0;col<imageH;col++)
     {
          D_in[row*imageW+col] = h_Src[row*imageW+col].x;
     }
}

init_phi(phi_in);
printf("Inisialisasi phi... SUKSES...\n");
save_to_file(phi_in, nama_txt_phi_awal);

kernel_gaussian(gaussian_matrix, sigma, kernel_size, "kernel.txt");
printf("Inisialisasi gaussian kernel... SUKSES...\n");
kernel_gaussian(gaussian_matrix_phi, sigma_phi, kernel_size_phi,
     "kernel_phi.txt");
printf("Inisialisasi gaussian kernel phi... SUKSES...\n");


HANDLE_ERROR( cudaEventRecord( start, 0 ) );
HANDLE_ERROR(cudaMemcpy(dev_D_in,D_in,imageSize,
     cudaMemcpyHostToDevice));
HANDLE_ERROR(cudaMemcpy(dev_phi_in,phi_in,imageSize,
     cudaMemcpyHostToDevice));
HANDLE_ERROR(cudaMemcpy(dev_gaussian_matrix,gaussian_matrix,kernel_
     size*kernel_size*sizeof(float), cudaMemcpyHostToDevice));
```

```
HANDLE_ERROR(cudaMemcpy(dev_gaussian_matrix_phi,gaussian_matrix_phi
      ,kernel_size_phi*kernel_size_phi*sizeof(float),
      cudaMemcpyHostToDevice));

for(int its=0;its<iterasi;its++)
{
      update_data_phi_GPU();
      c= c + 1;
      if(c % batas == 0)
      {
            if(status == 1)
            {
                  if((fp=fopen(nama_txt_time,"a")) == NULL)
                  {
                        printf("Cannot open file.\n");
                        exit(1);
                  }
                  HANDLE_ERROR(cudaEventRecord(stop,0));
                  HANDLE_ERROR(cudaEventSynchronize(stop));
                  HANDLE_ERROR(cudaEventElapsedTime(
                        &elapsedTime,start, stop));
                  fprintf (fp, "\n");
                  int b = imageW/x;
                  int t = (x*y);
                  fprintf (fp, "Time iterasi %d block = %d thread =
                        %d waktu = %0.2f detik", c, b, t,
                        (elapsedTime/1000));
                  fprintf (fp, "\n");
                  printf("%d \n", c);
            }
            else if(status == 2)
            {
                  HANDLE_ERROR(cudaMemcpy(phi_out,dev_phi_in,
                        imageSize, cudaMemcpyDeviceToHost));

                  sprintf(filename, "%s%d%s", one,c,type);
                  save_to_file(phi_out, filename);

                  phi_in = phi_out;
                  HANDLE_ERROR(cudaMemcpy(dev_phi_in,phi_in,
                        imageSize, cudaMemcpyHostToDevice));
            }
      }
}

HANDLE_ERROR(cudaMemcpy(diract_phi,dev_diract_phi,imageSize,
      cudaMemcpyDeviceToHost));
HANDLE_ERROR(cudaMemcpy(h_phi_in,dev_h_phi_in,imageSize,
      cudaMemcpyDeviceToHost));
HANDLE_ERROR(cudaMemcpy(D_out,dev_D_out, imageSize,
      cudaMemcpyDeviceToHost));
HANDLE_ERROR(cudaMemcpy(c1,dev_c1, imageSize,
      cudaMemcpyDeviceToHost));
HANDLE_ERROR(cudaMemcpy(c2,dev_c2, imageSize,
```

```
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(c1_out,dev_c1_out, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(c2_out,dev_c2_out, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(f1,dev_f1_out, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(f2,dev_f2_out, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(m1,dev_m1, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(m2,dev_m2, imageSize,
        cudaMemcpyDeviceToHost));
    HANDLE_ERROR(cudaMemcpy(phi_out,dev_phi_in, imageSize,
        cudaMemcpyDeviceToHost));

    HANDLE_ERROR(cudaEventRecord(stop,0));
    HANDLE_ERROR(cudaEventSynchronize(stop));
    HANDLE_ERROR(cudaEventElapsedTime(&elapsedTime,start, stop));
    HANDLE_ERROR(cudaEventDestroy(start));
    HANDLE_ERROR(cudaEventDestroy(stop));

    printf("SUKSES");

    free_GPU();
    free_CPU();
}

void konvolusi_CPU(float *pixel_data_in,float *pixel_data_out,
    float *kernel, int kernel_size, float sigma)
{
    float sum = 0.0;
    int half = (kernel_size-1)/2;
    int row_i=0;
    int col_j=0;

    for(row=0;row<imageW;row++)
    {
        for(col=0;col<imageH;col++)
        {
            sum = 0;
            for(int i=-half;i<=half;i++)
            {
                row_i = row + i;
                for(int j=-half;j<=half;j++)
                {
                    col_j = col + j;
                    if((col_j>=0)&&(col_j<imageH)
                        &&(row_i>=0)&&(row_i<imageW))
                    {
                        sum += (pixel_data_in[row_i*imageW+col_j]
                        *kernel[(i+half)*kernel_size+(j+half)]);
                    }
                }
```

```
                }
                pixel_data_out[row*imageW+col] = sum;
            }
        }
}

void pixel_data_h(float *pixel_data_in, float *pixel_data_out,
     float *h_phi_out, int status)
{
     for(row=0;row<imageW;row++)
     {
          for(col=0;col<imageH;col++)
          {
               if(status==1)
               {
                    pixel_data_out[row*imageW+col] =
                         pixel_data_in[row*imageW+col] *
                         h_phi_out[row*imageW+col];
                    h_phi_out[row*imageW+col] =
                         h_phi_out[row*imageW+col];
               }
               else
               {
                    pixel_data_out[row*imageW+col] =
                         pixel_data_in[row*imageW+col] * (1-
                         h_phi_out[row*imageW+col]);
                    h_phi_out[row*imageW+col] = 1 –
                         h_phi_out[row*imageW+col];
               }
          }
     }
}

void heaviside(float *phi, float *h_phi_out, float epsilon)
{
     for(int row=0;row<imageW;row++)
     {
          for(int col=0;col<imageH;col++)
          {
               h_phi_out[row*imageW+col] = 0.5 * (1 + (2/M_PI) *
                    atan(phi[row*imageW+col]/epsilon));
          }
     }
}

void m(float *pixel_data_in, float *pixel_data_out, float *h_phi_out,
     float *m, int status)
{
     float *f, *c;

     if((f=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
     {
          printf("me f %d", status);
     }
```

```
	if((c=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
	{
		printf("me c %d", status);
	}
	if((m=(float *)malloc(imageW*imageH*sizeof(float)))==NULL)
	{
		printf("me m %d", status);
	}

	pixel_data_h(pixel_data_in, pixel_data_out, h_phi_out, status);
	konvolusi_CPU(pixel_data_out,f, gaussian_matrix, kernel_size,
		sigma);
	konvolusi_CPU(h_phi_out,c, gaussian_matrix, kernel_size, sigma);

	for(row=0;row<imageW;row++)
	{
		for(col=0;col<imageH;col++)
		{
			if(status==1)
			{
				m1[row*imageW+col]=f[row*imageW+col]/
					c[row*imageW+col];
			}
			else
			{
				m2[row*imageW+col] = f[row*imageW+col]/
					c[row*imageW+col];
			}
		}
	}
}

void update_data_phi_CPU()
{

		delta_h(phi_in,diract_phi,epsilon);
		heaviside(phi_in,h_phi_in,epsilon);
		for(row=0;row<imageW;row++)
		{
			for(col=0;col<imageH;col++)
			{
				h_phi_out[row*imageW+col] =
					h_phi_in[row*imageW+col];
			}
		}
		m(D_in,D_out,h_phi_out,m1,1);
		m(D_in,D_out,h_phi_out,m2,2);
		for(row=0;row<imageW;row++)
		{
			for(col=0;col<imageH;col++)
			{
				N = row*imageW+col;
				phi_in[N] += timestep * diract_phi[N] *
					((D_in[N]-(m1[N]*h_phi_in[N])-(m2[N]*
```

```
                              (1-h_phi_in[N])))*(m1[N]-m2[N]));
                    }
               }
               konvolusi_CPU(phi_in, phi_in, gaussian_matrix_phi,
                    kernel_size_phi, sigma_phi);
}

void delta_h(float *phi, float *diract_phi, float epsilon)
{
     for(int row=0;row<imageW;row++)
     {
          for(int col=0;col<imageH;col++)
          {
               diract_phi[row*imageW+col] =
                    epsilon/(M_PI*((epsilon*epsilon)+
                    (phi[row*imageW+col]*phi[row*imageW+col])));
          }
     }
}

void save_to_file(float *data_array, const char *name)
{
     FILE *fp;
     if((fp=fopen(name,"w")) == NULL)
     {
          printf("Cannot open file.\n");
          exit(1);
     }
     for(row=0;row<imageW;row++)
     {
          for(col=0;col<imageH;col++)
          {
               /* write to file */
               fprintf(fp, " %4.4f ",
                    data_array[((imageW-1)-row)*imageW+col]);
               fprintf(fp,"\t");
          }
          fprintf(fp,"\n");
     }
     printf("Write file..  %s  SUKSES.....\n", name);
}



void init_phi(float *phi)
{
     for(row=0;row<imageW;row++)
     {
          for(col=0;col<imageH;col++)
          {
               if((row<15)||(col<15)||(row>(imageW-15))||
                    (col>(imageH-15)))
               {
                    phi[row*imageW+col] = 1;
```

```
                }
                else
                {
                        phi[row*imageW+col] = -1;
                }
        }
    }
}


void kernel_gaussian(float *kernel, float sigma, int kernel_size,
    const char *name)
{
    float r, s = 2.0 * sigma * sigma;
    float sum = 0.0;
    int half = (kernel_size-1)/2;

    FILE *fp;
    if((fp=fopen(name,"w")) == NULL)
    {
        printf("Cannot open file.\n");
        exit(1);
    }

    for(int x = (-1*half); x <= half; x++)
    {
        for(int y = (-1*half); y <= half; y++)
        {
            r = sqrt((float)(x*x + y*y));
            kernel[kernel_size*(x+half)+(y+half)] =
                (exp(-(r*r)/s))/(M_PI * s);
            sum += kernel[kernel_size*(x+half)+(y+half)];
        }
    }

    for(int i = 0; i < kernel_size; ++i)
    {
        for(int j = 0; j < kernel_size; ++j)
        {
            kernel[kernel_size*i+j] /= sum;
            fprintf(fp, " %4.4f ",  kernel[kernel_size*i+j]);
        }
        fprintf(fp,"\n");
    }
}

void free_CPU()
{
    free(D_in);
    free(D_out);
    free(h_phi_in);
    free(h_phi_out);
    free(m1);
    free(m2);
```

```
	free(f1);
	free(f2);
	free(c1);
	free(c2);
	free(c1_out);
	free(c2_out);
	free(diract_phi);
	free(phi);
	free(phi_in);
	free(phi_out);
}

void free_GPU()
{
	HANDLE_ERROR(cudaFree(dev_diract_phi));
	HANDLE_ERROR(cudaFree(dev_h_phi_in));
	HANDLE_ERROR(cudaFree(dev_h_phi_out));
	HANDLE_ERROR(cudaFree(dev_m1));
	HANDLE_ERROR(cudaFree(dev_m2));
	HANDLE_ERROR(cudaFree(dev_f1));
	HANDLE_ERROR(cudaFree(dev_f2));
	HANDLE_ERROR(cudaFree(dev_c1));
	HANDLE_ERROR(cudaFree(dev_c2));
	HANDLE_ERROR(cudaFree(dev_f1_out));
	HANDLE_ERROR(cudaFree(dev_f2_out));
	HANDLE_ERROR(cudaFree(dev_c1_out));
	HANDLE_ERROR(cudaFree(dev_c2_out));
	HANDLE_ERROR(cudaFree(dev_D_in));
	HANDLE_ERROR(cudaFree(dev_D_out));
	HANDLE_ERROR(cudaFree(dev_phi_in));
	HANDLE_ERROR(cudaFree(dev_phi_out));
	HANDLE_ERROR(cudaFree(dev_gaussian_matrix));
}

void save_result_to_txt()
{
	save_to_file(diract_phi, "diract_phi_GPU.txt");
	save_to_file(h_phi_in, "h_phi_GPU.txt");
	save_to_file(D_out, "D_out_GPU.txt");
	save_to_file(c1, "c1_GPU.txt");
	save_to_file(c2, "c2_GPU.txt");
	save_to_file(c1_out, "c1_out_GPU.txt");
	save_to_file(c2_out, "c2_out_GPU.txt");
	save_to_file(f1, "f1_GPU.txt");
	save_to_file(f2, "f2_GPU.txt");
	save_to_file(m1, "m1_GPU.txt");
	save_to_file(m2, "m2_GPU.txt");
	save_to_file(phi_out, "phi_out_GPU.txt");
}
```