BAB VI

KESIMPULAN DAN SARAN

VI.1. Kesimpulan

Dari pembahasan komputasi paralel untuk proses segmentasi citra pada video dengan menggunakan metode *level set*, dapat ditarik beberapa kesimpulan, yaitu:

1. Aplikasi segmentasi citra pada video dengan menggunakan metode *level set* yang berjalan pada CPU dan GPU dengan CUDA telah berhasil dikembangkan.
2. Percepatan maksimal terjadi pada video yang memiliki resolusi yang tinggi yaitu video *whiteball.mp4* dengan resolusi 480p. Semakin besar beban komputasi yang digunakan, maka akan semakin tinggi percepatannya.
3. Dengan mengggunakan GPU NVIDIA GeForce GTX 660 dapat mempercepat proses komputasi hingga 16 kali dibandingkan dengan CPU i7-3770K untuk video dengan resolusi 360p dengan ukuran *frame*: 360x360 dan dengan total *frame* ialah 342.
4. Dengan mengggunakan GPU NVIDIA GeForce GTX 660 dapat mempercepat proses komputasi hingga 17 kali dibandingkan dengan CPU i7-3770K untuk video dengan resolusi 420p dengan ukuran *frame*: 854x420 dan dengan total *frame* ialah 53.

VI.2. Saran

Beberapa saran dari penulis untuk penelitian bagi segmentasi citra pada video dengan menggunakan metode *level set* secara paralel:

1. Program dapat dioptimalkan dengan mengubah penggunaan *global memory* menjadi *texture memory* atau *shared memory*

2. Program dapat dioptimalkan dengan membuat Inisialisasi *mask* setiap *frame* video berdasarkan letak obyek yang akan disegmentasi.
3. Program dapat dikembangkan dengan menambahkan GUI, agar pengoperasian menjadi lebih mudah
4. Untuk Penelitian selanjutnya, Program dapat dikembangkan untuk mengolah video dengan citra biomedis.

DAFTAR PUSTAKA

Al-Ayyoub, M. et al., 2015. A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. *Springer-Verlag,* Issue 8, pp. 3149-3162.

Anon., n.d. *Youtube.* [Online]
Available at: https://www.youtube.com/watch?v=GYasCmb6agE
[Accessed 20 5 2016].

Anon., n.d. *Youtube.* [Online]
Available at: https://www.youtube.com/watch?v=oORgRSWdimI
[Accessed 17 June 2016].

Barrionuevo, M., Lopresti, M., Miranda, N. & iccoli, F., 2015. Solving Big-Data Problem with GPU: The Network Traffic Analysis. *JCS&T,* Volume XV, pp. 30-39.

Chen, J., Grossman, M. & McKercher, T., 2014. *Professional CUDA C Programming.* Indianapolis: John Wiley & Sons, Inc.

Chien, S. Y. & Chen, L. G., 2011. Reconfigurable Morphological Image Processing Accelerator for Video Object Segmentation. *Journal of Signal Processing Systems,* Issue 1, pp. 77-96.

Chi, J. et al., 2011. GPU-Accelerated FDTD Modeling of Radio-Frequency Field–Tissue Interactions in High-Field MRI. *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING,* pp. 1789-1796.

Febrihani, L., 2014. *Segmentasi Citra Menggunakan Level Set Untuk Active Contour Berbasis Parallel GPU CUDA,* Yogyakarta: Universitas Atma Jaya Yogyakarta.

Ghorpade, J., Parande, J., Kulkarni, M. & Bawaskar, A., 2012. GPU Processing in CUDA Architecture. *Advanced Computing: An International Journal (ACIJ),* Volume III, pp. 105-120.

Gross, J., Janke, W. & Bachmann, M., 2011. Massively Parallelized Replica-Exchange Simulations of Polymers on GPUs. *Computer Physics Communications,* pp. 1638-1644.

Jackson, A., 2009. *A PARALLEL ALGORITHM FOR FAST EDGE DETECTION ON THE GRAPHICS PROCESSING UNIT,* Washington: The Faculty of the Department of Computer Science Washington and Lee University.

Lefohn, A., Kniss, J., Hansen, C. & Whitaker, R., 2004. A Streaming Narrow-Band Algorithm: Interactive Computation and Visualization of Level Sets. *Institute of Electrical and Electronics Engineers,* X(4), pp. 0-433.

Michael, K., Andrew, W. & Demetri, T., 1988. Snakes: Active contour models. *International Journal of Computer Vision,* I(4), pp. 321-331.

Mostofi, H. & College, K., 2009. *Fast Level Set Segmentation of Biomedical Image Using Graphics Processing Units,* Oxford: University of Oxford.

Nan, L. B., Chui, C. K., Chang, S. & Ong, S. H., 2011. Integrating Spatial Fuzzy Clustering with Level Set Method for Automated Medical Image Segmentation. *Computers in Biology and Medicine,* pp. 1-10.

Osher, S. & Fedkiw, R., 2003. Level Set Method and Dynamic Implicit Surface. *Springer,* Volume 153.

Panca, I. G. P. A. W., 2015. *Komputasi Paralel Berbasis GPU CUDA Untuk Pengembangan Image Inpainting dengan Metode Perona-Malik,* Yogyakarta: Universitas Atma Jaya Yogyakarta.

Prosser, N. T., 2010. *Medical Image Segmentation using GPU-Accelerated Variational Level Set Methods,* New York: Rochester Institute of Technology.

Salomon, C. & Breckon, T., 2010. Fundamentals of Digital Image Processing (A Practical Approach with Examples in Matlab). pp. 1-19.

Samson, C., Laure, B.-F., Aubert, G. & Zerubia, J., 2000. A Level Set Model for Image Classification. *International Journal of Computer Vision,* Issue 3, pp. 187-197.

Sanders, J. & Kandrot, E., 201. *CUDA By Example (An Introduction to General-Purpose GPU Programming).* Boston: Addison-Wesley.

Spiechowicz, J., Kostur, M. & Machura, L., 2014. GPU Accelerated Monte Carlo Simulation of Brownian Motor Dynamics with CUDA. *Computer Physics Communications,* pp. 140-149.

Yao & Chen, T., 2010. A Level Set Method Based on the Bayesian Risk for Medical Image Segmentation. *Pattern Recognition,* pp. 3699-3711.

Yuan, Y. & He, C., 2013. Variational Level Set Method for Image Segmentation Based on Both L2 and Sobolev Gradient. *Nonlinear Analysis: Real World Applications,* Volume XIII, pp. 959-956.

Zhang, K., Song, H. & Zhang, L., 2010. Active Contours Driven by Local Image Fitting Energy. *Pattern Recognition,* Issue 4, pp. 1199-1206.

LAMPIRAN

```
//--Library CUDA
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

//--Library C++
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <Windows.h>
#include <fstream>
#include <iostream>

//--Libraty OpenCV
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>


using namespace std;
using namespace cv;

#define EPSILON             35
//#define THRESHOLD   158
//#define ITTERATION   450

#define ALPHA        0.009
#define DT                  0.25

//--Variabel Global CPU dan GPU
                                                                double *phi, *D;
int imageW, imageH, N, iterasi, inisialisasi, thres;

//Variabel device (GPU)
double *d_dx, *d_dxplus, *d_dxminus, *d_dxplusy, *d_dxminusy, *d_maxdxplus,
*d_maxminusdxminus, *d_mindxplus, *d_minminusdxminus;
double *d_dy, *d_dyplus, *d_dyminus, *d_dyplusx, *d_dyminusx, *d_maxdyplus,
*d_maxminusdyminus, *d_mindyplus, *d_minminusdyminus;
double *d_gradphimax, *d_gradphimin;
double *d_nplusx, *d_nplusy, *d_nminusx, *d_nminusy;
double *d_curvature, *d_F, *d_gradphi, *d_phi, *d_phi_new, *d_D;

//Variabel Host CPU
double *h_dx, *h_dxplus, *h_dxminus, *h_dxplusy, *h_dxminusy, *h_maxdxplus,
*h_maxminusdxminus, *h_mindxplus, *h_minminusdxminus;
double *h_dy, *h_dyplus, *h_dyminus, *h_dyplusx, *h_dyminusx, *h_maxdyplus,
*h_maxminusdyminus, *h_mindyplus, *h_minminusdyminus;
double *h_gradphimax, *h_gradphimin, *h_nplusx, *h_nplusy, *h_nminusx,
*h_nminusy, *h_curvature, *h_F, *h_gradphi;
```

```
vector<Mat> Images_Frame;
vector<Mat> Phi_Frame_GPU;
vector<Mat> Phi_Frame_CPU;


//Deklarasi Prosedur dan Fungsi
__global__ void dx(double *d_phi, double *d_dx, int rows, int col);
__global__ void dxplus(double *d_phi, double *d_dxplus, int rows, int col);
__global__ void dxminus(double *d_phi, double *d_dxminus, int rows, int col);
__global__ void dxplusy(double *d_phi, double *d_dxplusy, int rows, int col);
__global__ void dxminusy(double *d_phi, double *d_dxminusy, int rows, int col);
__global__ void maxdxplus(double *d_dxplus, double *d_maxdxplus, int rows, int
col);
__global__ void maxminusdxminus(double *d_dxminus, double *d_maxminusdxminus,
int rows, int col);
__global__ void mindxplus(double *d_dxplus, double *d_mindxplus, int rows, int
col);
__global__ void minminusdxminus(double *d_dxminus, double *d_minminusdxminus,
int rows, int col);

__global__ void dy(double *d_phi, double *d_dy, int rows, int col);
__global__ void dyplus(double *d_phi, double *d_dyplus, int rows, int col);
__global__ void dyminus(double *d_phi, double *d_dyminus, int rows, int col);
__global__ void dyplusx(double *d_phi, double *d_dyplusx, int rows, int col);
__global__ void dyminusx(double *d_phi, double *d_dyminusx, int rows, int col);
__global__ void maxdyplus(double *dyplus, double *d_maxdyplus, int rows, int
col);
__global__ void maxminusdyminus(double *d_dyminus, double *d_maxminusdyminus,
int rows, int col);
__global__ void mindyplus(double *dyplus, double *d_mindyplus, int rows, int
col);
__global__ void minminusdyminus(double *d_dyminus, double *d_minminusdyminus,
int rows, int col);

__global__ void gradphimax(double *d_gradphimax, double *d_maxdxplus, double
*d_maxminusdxminus, double *d_maxdyplus, double *d_maxminusdyminus, int rows,
int col);
__global__ void gradphimin(double *d_gradphimin, double *d_mindxplus, double
*d_minminusdxminus, double *d_mindyplus, double *d_minminusdyminus, int rows,
int col);

__global__ void nplusx(double *d_nplusx, double *d_dxplus, double *d_dyplusx,
double *d_dy, float eps, int rows, int col);
__global__ void nplusy(double *d_nplusy, double *d_dyplus, double *d_dxplusy,
double *d_dx, float eps, int rows, int col);
__global__ void nminusx(double *d_nminusx, double *d_dxminus, double
*d_dyminusx, double *d_dy, float eps, int rows, int col);
__global__ void nminusy(double *d_nminusy, double *d_dyminus, double
*d_dxminusy, double *d_dx, float eps, int rows, int col);

__global__ void curvature(double *d_curvature, double *d_nplusx, double
*d_nminusx, double *d_nplusy, double *d_nminusy, int rows, int col);
```

```
__global__ void F(double *d_F, double *d_D,double *d_curvature, double aplha,
int rows, int col);
__global__ void gradphi(double *d_gradphi, double *d_F, double *d_gradphimax,
double *d_gradphimin, int rows, int col);
__global__ void update_phi(double *d_phi_new, double *d_phi_old, double *d_F,
double *d_gradphi, double dt, int rows, int col);
__global__ void copy_data(double *d_new, double *d_old, int rows, int col);


void ambil_citra(Mat imagesrc, Mat *imagedst, Mat mask, int imageH, int
imageW);
void init_phi(Mat image, int init);
void readvideo(char nama[30], double *TotalFrame, int *width, int *height,
double *FPS);
void displayVideo(vector<Mat> videoframes);
void WriteVideoGPU(double TotalFrame, int width, int height, double FPS);
void WriteVideoCPU(double TotalFrame, int width, int height, double FPS);
void InisialisaiD(Mat tempD);
void Alokasi_Memori_GPU();
void Alokasi_Memori_CPU();
void save_to_file(double *data, const char *name);
void Free_Memory_GPU();
void Free_Memory_CPU();
void update_phi();
void update_phi_GPU();
void Image_Segmentation_CPU(int iterasi);
void Image_Segmentation_GPU(int iterasi);
void SegmentasiVideo_LevelSet_CPU();
void SegmentasiVideo_LevelSet_GPU();


int main()
{
	char nama_video[30];
	int masukan;
	//=====Variabel atribut video=====//
	int  width = 0, height = 0;
	double FPS = 0, totalframes = 0;
	N = 0;

	do
	{
		system("cls");
		printf("\t\t\tLevel Set Video Segmentation\n\n");

	printf("*****************************************************************
*****\n");

		printf("\n[CPU Specification]\n");
		printf("NAME\t: Intel<R> Core<TM> i7-3770K CPU @ 3.50GHz\n");
		printf("MEMORY\t: 16384 MB\n\n");

		printf("\n[GPU Specification]\n");
```

```
            printf("NAME\t: NVIDIA GeForce GTX 660\n");
            printf("MEMORY\t: 2021 MB\n\n");


      printf("**************************************************************
*****\n");

            printf("\t\t<--------------MENU-------------->");
            printf("\n\t\t<---  1. Input Video          --->");
            printf("\n\t\t<---  2. Segmentasi Video CPU  -->");
            printf("\n\t\t<---  3. Segmentasi Video GPU  -->");
            printf("\n\t\t<---  0. Keluar                -->");

            printf("\n\n##-----------------------------------------------------
---------------\n");
            printf("Masukkan Menu : ");scanf("%d",&masukan);
            printf("##-----------------------------------------------------
-----------\n");

            switch (masukan)
            {
            case 1:       //printf("\n\n=========MEMBACA VIDEO==========\n");
                          printf("\nMasukkan Nama Video\t:
");scanf("%s",&nama_video);
                          readvideo(nama_video,&totalframes, &width, &height,
&FPS);//read video and input frame to Images_Frame

                          imageW = width;
                          imageH = height;
                          N = imageH * imageW;

                          //displayVideo(Images_Frame);

                          getch();
                          break;

            case 2: if(N==0)
                          {
                                printf("\nSilahkan Pilih Video Terlebih
Dahulu\n");
                          }
                          else
                          {
                                printf("\n\n=========SEGMENTASI CITRA PADA
VIDEO (CPU)==========\n");
                                do
                                {
                                      printf("Masukkan Jumlah Iterasi yang
Diinginkan\t: ");scanf("%d",&iterasi);
                                } while (iterasi<1);
                                do
                                {
```

```
                                        printf("Masukkan Inisialisasi
Segmentasi\t: ");scanf("%d",&inisialisasi);
                                    } while (inisialisasi<1);
                                    do
                                    {
                                        printf("Masukkan Nilai Threshold\t\t:
");scanf("%d",&thres);
                                    } while (thres<0);
                                    printf("\n");
                                    SegmentasiVideo_LevelSet_CPU();
                                    WriteVideoCPU(totalframes, width, height,
FPS);
                                    printf("\nVideo telah tersimpan\n");

                                }
                                getch();
                                break;

                    case 3:     if(N==0)
                                {
                                    printf("\nSilahkan Pilih Video Terlebih
Dahulu\n");
                                }
                                else
                                {
                                    printf("\n\n=========SEGMENTASI CITRA PADA
VIDEO (GPU)==========\n");
                                    do
                                    {
                                        printf("Masukkan Jumlah Iterasi yang
Diinginkan\t: ");scanf("%d",&iterasi);
                                    } while (iterasi<1);
                                    do
                                    {
                                        printf("Masukkan Inisialisasi
Segmentasi\t: ");scanf("%d",&inisialisasi);
                                    } while (inisialisasi<1);
                                    do
                                    {
                                        printf("Masukkan Nilai Threshold\t\t:
");scanf("%d",&thres);
                                    } while (thres<0);
                                    printf("\n\n");
                                    SegmentasiVideo_LevelSet_GPU();
                                    WriteVideoGPU(totalframes, width, height,
FPS);
                                    printf("\nVideo telah tersimpan\n");
                                }
                                getch();
                                break;

                    }
```

```
	} while (masukan != 0);

	waitKey(0);

	return 0;
}

////////////////////////////////dx//////////////////////////////////////////
//
__global__ void dx(double *d_phi, double *d_dx, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;
	double left, right;

	if(x < col && y<rows)
	{
		if(x == 0)
		{
			left = 0.0;
			right = d_phi[index + 1];
		}

		else if(x == col - 1)
		{
			left = d_phi[index - 1];
			right = 0.0;
		}
		else
		{
			left = d_phi[index - 1];
			right = d_phi[index + 1];
		}

		d_dx[index] = (right - left)/2.0;
	}

	__syncthreads();

}

__global__ void dxplus(double *d_phi, double *d_dxplus, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;
	double right;

	if(x < col && y<rows)
	{
```

```
		if(x == col - 1)
		{
			right = 0;
		}
		else
		{
			right = d_phi[index + 1];
		}

		d_dxplus[index] = right - d_phi[index];
	}

	__syncthreads();

}

__global__ void dxminus(double *d_phi, double *d_dxminus, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;
	double left;

	if(x < col && y<rows)
	{
		if(x == 0)
		{
			left = 0;
		}
		else
		{
			left = d_phi[index - 1];
		}

		d_dxminus[index] = d_phi[index] - left;

	}

	__syncthreads();

}

__global__ void dxplusy(double *d_phi, double *d_dxplusy, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	double r_bottom, l_bottom;

	if(x < col && y<rows)
```

```
	{
		if(y == rows - 1)
		{
			r_bottom = 0;
			l_bottom = 0;
		}

		else if(x==0)
		{
			l_bottom = 0;
			r_bottom = d_phi[index + 1 + col];
		}
		else if (x == col - 1)
		{
			l_bottom = d_phi[index - 1 + col];
			r_bottom = 0;
		}
		else
		{
			l_bottom = d_phi[index - 1 + col];
			r_bottom = d_phi[index + 1 + col];
		}

		d_dxplusy[index] = (r_bottom - l_bottom)/2;

	}

	__syncthreads();
}

__global__ void dxminusy(double *d_phi, double *d_dxminusy, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	double r_top, l_top;

	if(x < col && y<rows)
	{
		if(y == 0)
		{
			r_top = 0;
			l_top = 0;
		}

		else if(x == 0)
		{
			l_top = 0;
			r_top = d_phi[index + 1 - col];
		}
		else if (x == col - 1)
```

```
		{
			l_top = d_phi[index - 1 - col];
			r_top = 0;
		}
		else
		{
			l_top = d_phi[index - 1 - col];
			r_top = d_phi[index + 1 - col];
		}

		d_dxminusy[index] = (r_top - l_top)/2;
	}

	__syncthreads();

}

__global__ void maxdxplus(double *d_dxplus, double *d_maxdxplus, int rows, int
col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if (d_dxplus[index] < 0)
		{
			d_maxdxplus[index] = 0;
		}
		else
		{
			d_maxdxplus[index] = (d_dxplus[index] * d_dxplus[index]);
		}

	}

	__syncthreads();
}

__global__ void maxminusdxminus(double *d_dxminus, double *d_maxminusdxminus,
int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;


	if(x < col && y<rows)
	{
		if (-(d_dxminus[index]) < 0)
```

```
		{
			d_maxminusdxminus[index] = 0;
		}
		else
		{
			d_maxminusdxminus[index] = (d_dxminus[index] *
d_dxminus[index]);
		}
	}

	__syncthreads();
}

__global__ void mindxplus(double *d_dxplus, double *d_mindxplus, int rows, int
col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if (d_dxplus[index] > 0)
		{
			d_mindxplus[index] = 0;
		}
		else
		{
			d_mindxplus[index] = (d_dxplus[index] * d_dxplus[index]);
		}
	}

	__syncthreads();

}

__global__ void minminusdxminus(double *d_dxminus, double *d_minminusdxminus,
int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if (-(d_dxminus[index]) > 0)
		{
			d_minminusdxminus[index] = 0;
		}
		else
		{
```

```
                    d_minminusdxminus[index] = (d_dxminus[index] *
d_dxminus[index]);
             }
      }

      __syncthreads();
}

//////////////////////////DY//////////////////////////

__global__ void dy(double *d_phi, double *d_dy, int rows, int col)
{
      int x =threadIdx.x + blockIdx.x*blockDim.x;
      int y =threadIdx.y + blockIdx.y*blockDim.y;

      int index = x+ y*col;

      double top, bottom;

      if(x < col && y<rows)
      {
            if(y == 0)
            {
                  top = 0;
                  bottom = d_phi[index + col];
            }

            else if(y == rows - 1)
            {
                  top = d_phi[index - col];
                  bottom = 0;
            }
            else
            {
                  top = d_phi[index - col];
                  bottom = d_phi[index + col];
            }

            d_dy[index] = (bottom - top)/2;
      }

      __syncthreads();

}

__global__ void dyplus(double *d_phi, double *d_dyplus, int rows, int col)
{
      int x =threadIdx.x + blockIdx.x*blockDim.x;
      int y =threadIdx.y + blockIdx.y*blockDim.y;

      int index = x+ y*col;

      double bottom;
```

```
	if(x < col && y<rows)
	{
		if(y == rows - 1)
		{
			bottom = 0;
		}
		else
		{
			bottom = d_phi[index + col];
		}

		d_dyplus[index] = bottom - d_phi[index];
	}

	__syncthreads();

}

__global__ void dyminus(double *d_phi, double *d_dyminus, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	double top;

	if(x < col && y<rows)
	{

		if(y == 0)
		{
			top = 0;
		}
		else
		{
			top = d_phi[index - col];
		}

		d_dyminus[index] = d_phi[index] - top;
	}

	__syncthreads();

}

__global__ void dyplusx(double *d_phi, double *d_dyplusx, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;
```

```
	double t_right, b_right;

	if(x < col && y<rows)
	{

		if(x == col - 1)
		{
			t_right = 0;
			b_right = 0;
		}
		else if(y == 0)
		{
			t_right = 0;
			b_right = d_phi[index + 1 + col];
		}
		else if(y == rows - 1)
		{
			t_right = d_phi[index + 1 - col];
			b_right = 0;
		}
		else
		{
			t_right = d_phi[index + 1 - col];
			b_right = d_phi[index + 1 + col];
		}

		d_dyplusx[index] = (b_right - t_right)/2;
	}

	__syncthreads();
}

__global__ void dyminusx(double *d_phi, double *d_dyminusx, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	double t_left, b_left;

	if(x < col && y<rows)
	{
		if(x == 0)
		{
			t_left = 0;
			b_left = 0;
		}
		else if(y == 0)
		{
			t_left = 0;
			b_left = d_phi[index - 1 + col];
```

```
		}
		else if(y == rows - 1)
		{
			t_left = d_phi[index - 1 - col];
			b_left = 0;
		}
		else
		{
			t_left = d_phi[index - 1 - col];
			b_left = d_phi[index - 1 + col];
		}

		d_dyminusx[index] = (b_left - t_left)/2;
	}

	__syncthreads();

}

__global__ void maxdyplus(double *dyplus, double *d_maxdyplus, int rows, int
col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if(dyplus[index] < 0)
		{
			d_maxdyplus[index] = 0;
		}
		else
		{
			d_maxdyplus[index] = (dyplus[index] * dyplus[index]);
		}
	}

	__syncthreads();
}

__global__ void maxminusdyminus(double *d_dyminus, double *d_maxminusdyminus,
int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if(-(d_dyminus[index]) < 0)
```

```
		{
			d_maxminusdyminus[index] = 0;
		}
		else
		{
			d_maxminusdyminus[index] = (d_dyminus[index] *
d_dyminus[index]);
		}
	}

	__syncthreads();

}

__global__ void mindyplus(double *dyplus, double *d_mindyplus, int rows, int
col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if(dyplus[index] > 0)
		{
			d_mindyplus[index] = 0;
		}
		else
		{
			d_mindyplus[index] = (dyplus[index] * dyplus[index]);
		}
	}

	__syncthreads();
}

__global__ void minminusdyminus(double *d_dyminus, double *d_minminusdyminus,
int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if(-(d_dyminus[index]) > 0)
		{
			d_minminusdyminus[index] = 0;
		}
		else
		{
```

```
                    d_minminusdyminus[index] = (d_dyminus[index] *
d_dyminus[index]);
              }
       }

       __syncthreads();

}


////////////////////////////////gradphimaxmin/////////////////////////////////////
//
__global__ void gradphimax(double *d_gradphimax, double *d_maxdxplus, double
*d_maxminusdxminus, double *d_maxdyplus, double *d_maxminusdyminus, int rows,
int col)
{
       int x =threadIdx.x + blockIdx.x*blockDim.x;
       int y =threadIdx.y + blockIdx.y*blockDim.y;

       int index = x+ y*col;

       if(x < col && y<rows)
       {
              d_gradphimax[index] =
sqrt((sqrt(d_maxdxplus[index]+d_maxminusdxminus[index]))*(sqrt(d_maxdxplus[inde
x]+d_maxminusdxminus[index])) +
(sqrt(d_maxdyplus[index]+d_maxminusdyminus[index]))*(sqrt(d_maxdyplus[index]+d_
maxminusdyminus[index])));
       }

       __syncthreads();
}

__global__ void gradphimin(double *d_gradphimin, double *d_mindxplus, double
*d_minminusdxminus, double *d_mindyplus, double *d_minminusdyminus,int rows,
int col)
{
       int x =threadIdx.x + blockIdx.x*blockDim.x;
       int y =threadIdx.y + blockIdx.y*blockDim.y;

       int index = x+ y*col;

       if(x < col && y<rows)
       {
              d_gradphimin[index] =
sqrt((sqrt(d_mindxplus[index]+d_minminusdxminus[index]))*(sqrt(d_mindxplus[inde
x]+d_minminusdxminus[index])) +
(sqrt(d_mindyplus[index]+d_minminusdyminus[index]))*(sqrt(d_mindyplus[index]+d_
minminusdyminus[index])));
       }

       __syncthreads();
```

```
}


//////////////////////n/////////////////////////////////////
__global__ void nplusx(double *d_nplusx, double *d_dxplus, double *d_dyplusx,
double *d_dy, float eps, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_nplusx[index] = d_dxplus[index]/sqrt(eps +
(d_dxplus[index]*d_dxplus[index]) +
((d_dyplusx[index]+d_dy[index])*(d_dyplusx[index]+d_dy[index])*0.25));
	}

	__syncthreads();

}

__global__ void nplusy(double *d_nplusy, double *d_dyplus, double *d_dxplusy,
double *d_dx, float eps, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_nplusy[index] = d_dyplus[index]/sqrt(eps +
(d_dyplus[index]*d_dyplus[index]) +
((d_dxplusy[index]+d_dx[index])*(d_dxplusy[index]+d_dx[index])*0.25));
	}

	__syncthreads();

}

__global__ void nminusx(double *d_nminusx, double *d_dxminus, double
*d_dyminusx, double *d_dy, float eps, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
```

```
		d_nminusx[index] = d_dxminus[index]/sqrt(eps +
(d_dxminus[index]*d_dxminus[index]) +
((d_dyminusx[index]+d_dy[index])*(d_dyminusx[index]+d_dy[index])*0.25));
	}

	__syncthreads();
}

__global__ void nminusy(double *d_nminusy, double *d_dyminus, double
*d_dxminusy, double *d_dx, float eps, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_nminusy[index] = d_dyminus[index]/sqrt(eps +
(d_dyminus[index]*d_dyminus[index]) +
((d_dxminusy[index]+d_dx[index])*(d_dxminusy[index]+d_dx[index])*0.25));
	}

	__syncthreads();
}

//curvature
__global__ void curvature(double *d_curvature, double *d_nplusx, double
*d_nminusx, double *d_nplusy, double *d_nminusy, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_curvature[index] = ((d_nplusx[index]-
d_nminusx[index])+(d_nplusy[index]-d_nminusy[index]))/2;
	}

	__syncthreads();
}

///F
__global__ void F(double *d_F, double *d_D,double *d_curvature, double aplha,
int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;
```

```
	if(x < col && y<rows)
	{
		d_F[index] = -(aplha * d_D[index]) + ((1 - aplha) *
d_curvature[index]);
	}
	__syncthreads();

}

//gradphi
__global__ void gradphi(double *d_gradphi, double *d_F, double *d_gradphimax,
double *d_gradphimin, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		if(d_F[index] > 0)
		{
			d_gradphi[index] = d_gradphimax[index];
		}
		else
		{
			d_gradphi[index] = d_gradphimin[index];
		}
	}

	__syncthreads();

}

//update phi
__global__ void update_phi(double *d_phi_new, double *d_phi_old, double *d_F,
double *d_gradphi, double dt, int rows, int col)
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_phi_new[index] = d_phi_old[index] + dt * d_F[index] *
d_gradphi[index];
	}

	__syncthreads();
}

__global__ void copy_data(double *d_new, double *d_old, int rows, int col)
```

```
{
	int x =threadIdx.x + blockIdx.x*blockDim.x;
	int y =threadIdx.y + blockIdx.y*blockDim.y;

	int index = x+ y*col;

	if(x < col && y<rows)
	{
		d_new[index] = d_old[index];
	}

	__syncthreads();

}



void ambil_citra(Mat imagesrc, Mat *imagedst, Mat mask, int imageH, int imageW)
{
	//Konversi citra 8UC1 ke 64FC1 agar piksel pada citra dapat diolah
	imagesrc.convertTo(imagesrc, CV_64FC1);
	(*imagedst).convertTo((*imagedst), CV_64FC1);
	mask.convertTo(mask, CV_64FC1);

	//Inisialisasi citra hasil
	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{
			(*imagedst).at<double>(x,y) = 255.0;

		}
	}

	//Proses pengambilan citra asli berdasarkan mask
	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{
			if(mask.at<double>(x,y) != 255.0)
			{
				(*imagedst).at<double>(x,y) =
imagesrc.at<double>(x,y);
			}

		}
	}

	//konversi citra tipe 64FC1 ke 8UC1
	imagesrc.convertTo(imagesrc, CV_8UC1);
	(*imagedst).convertTo((*imagedst), CV_8UC1);
	mask.convertTo(mask, CV_8UC1);
	//imshow("ambil",imagedst);
```

```
}

void init_phi(Mat image, int init)
{

	double *mask;
	mask = (double *)malloc(imageH*imageW*sizeof(double));

	Mat mask_image;
	mask_image = image.clone();

	//--Konversi citra dari tipe uchar 8 bit dan 1 channel ke tipe float 64
bit 1 channel
	mask_image.convertTo(mask_image,CV_64FC1);

	//--Mengubah nilai piksel dari citra
	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{

			if(y>=(imageW/init) && y<(imageW/init)*(init-1) &&
x>=(imageH/init) && x<(imageH/init)*(init-1))
			{
				mask_image.at<double>(x,y) = 0.0;
			}
			else
			{
				mask_image.at<double>(x,y) = 255.0;
			}

		}
	}

	//--Konversi Citra 2 Dimensi ke Array 1 Dimensi
	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{
			int i1 = y + x * imageW;

			mask[i1] = mask_image.at<double>(x,y);
		}
	}

	//--Konversi citra dari tipe float 64 bit 1 channel ke tipe uchar 8 bit
dan 1 channel
	mask_image.convertTo(mask_image,CV_8UC1);


	for(int i = 0; i < imageH; i++)
	{
		for (int j = 0; j < imageW; j++)
```

```
		{
			int i1 = i*imageW+j;

			if(mask[i1] > 0)
			{
				phi[i1] = mask[i1];//0.5*sqrt(abs(mask[i1]));
			}
			else
			{
				phi[i1] = -mask[i1];//c -0.5*sqrt(abs(mask[i1]));
			}
		}
	}

	//--Konversi citra dari tipe uchar 8 bit dan 1 channel ke tipe float 64
bit 1 channel
	mask_image.convertTo(mask_image, CV_64FC1);

	//--Konversi Array 1 Dimensi ke Citra 2 Dimensi
	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{
			int i1 = y + x * imageW;

			mask_image.at<double>(x,y) = phi[i1];
		}
	}

	//--Konversi citra dari tipe float 64 bit 1 channel ke tipe uchar 8 bit
dan 1 channel
	mask_image.convertTo(mask_image,CV_8UC1);
	//imshow("phi",mask_image);

	free(mask);

}

void readvideo(char nama[30], double *TotalFrame, int *width, int *height,
double *FPS)
{
	Mat temp_gray;

	//VideoCapture cap("video.mp4");
	VideoCapture cap(nama);
	if(!cap.isOpened())
	{
		printf("Video tidak dapat diinput (video tidak ada didalam
project)\n");
		//exit(-1);
		 *TotalFrame = 0;
		 *width = 0;
		 *height = 0;
```

```
                *FPS = 0;
        }
        else
        {
               *TotalFrame = cap.get(CV_CAP_PROP_FRAME_COUNT);

                // Get the width/height and the FPS of the vide
                *width = static_cast<int>(cap.get(CV_CAP_PROP_FRAME_WIDTH));
                *height = static_cast<int>(cap.get(CV_CAP_PROP_FRAME_HEIGHT));
                *FPS = cap.get(CV_CAP_PROP_FPS);


                cv::Mat image, img_clone;

                while(true) {
                       cap >> image;
                       if(image.empty()) {
                              //printf("Can't read frames from your camera\n");
                              break;
                       }
                       img_clone = image.clone();
                      cvtColor(img_clone, temp_gray, CV_RGB2GRAY);
                      Images_Frame.push_back(temp_gray.clone());

                }

                printf("\n\nDeskripsi Video: \n");
                printf("Nama Video\t: %s\n\n",nama);
                printf("Ukuran Frame\nWidth \t\t:%d\nHeight \t\t:%d\n",
(*width), (*height));
                printf("Total Frame \t:%lf\nFPS \t\t:%lf\n", (*TotalFrame),
(*FPS));

        }

}

void displayVideo(vector<Mat> videoframes)
{
        Mat tempframe;
        //int ind = 0;

        while(!videoframes.empty()) //Show the image captured in the window and
repeat
    {
                //printf("%d\n",ind);
         tempframe = videoframes.back();
         imshow("video", tempframe);
         videoframes.pop_back();
         waitKey(20); // waits to display frame
                //ind++;
    }
}
```

```
void WriteVideoGPU(double TotalFrame, int width, int height, double FPS)
{
	Mat tempframe, newss;

	// Open a video file for writing (the MP4V codec works on OS X and
Windows)
	//VideoWriter out("output.mov", CV_FOURCC('m','p', '4', 'v'), FPS,
Size(width, height));
	VideoWriter out("output_gpu.mov", CV_FOURCC('m','p', '4', 'v'), FPS,
Size(width, height));
     if(!out.isOpened()) {
	   printf("Error! Unable to open video file for output.");
	   exit(-1);
     }

	 for (vector<Mat>::iterator iter = Phi_Frame_GPU.begin(); iter !=
Phi_Frame_GPU.end(); iter++)
	 {
		tempframe = *iter;
		cvtColor(tempframe,newss,CV_GRAY2RGB);
		out << newss;
	 }


}

void InisialisaiD(Mat tempD)
{
	Mat initD;

	initD = tempD.clone();
	initD.convertTo(initD, CV_64FC1);

	for (int y = 0; y < imageW; y++)
	{
		for (int x = 0; x < imageH; x++)
		{
			int i1 = y + x * imageW;

			D[i1] = initD.at<double>(x,y);
		}
	}

	for (int i = 0; i < N; i++)
	{
		D[i] = EPSILON - abs(D[i] - thres);
	}
}

void Alokasi_Memori_GPU()
{
	phi = (double *)malloc(imageH*imageW*sizeof(double));
```

```
	D = (double *)malloc(imageH*imageW*sizeof(double));

	//Alokasi Cuda Memory
	cudaMalloc((void**)&d_dx,N*sizeof(double));
	cudaMalloc((void**)&d_dxplus,N*sizeof(double));
	cudaMalloc((void**)&d_dxminus,N*sizeof(double));
	cudaMalloc((void**)&d_dxplusy,N*sizeof(double));
	cudaMalloc((void**)&d_dxminusy,N*sizeof(double));
	cudaMalloc((void**)&d_maxdxplus,N*sizeof(double));
	cudaMalloc((void**)&d_maxminusdxminus,N*sizeof(double));
	cudaMalloc((void**)&d_mindxplus,N*sizeof(double));
	cudaMalloc((void**)&d_minminusdxminus,N*sizeof(double));

	cudaMalloc((void**)&d_dy,N*sizeof(double));
	cudaMalloc((void**)&d_dyplus,N*sizeof(double));
	cudaMalloc((void**)&d_dyminus,N*sizeof(double));
	cudaMalloc((void**)&d_dyplusx,N*sizeof(double));
	cudaMalloc((void**)&d_dyminusx,N*sizeof(double));
	cudaMalloc((void**)&d_maxdyplus,N*sizeof(double));
	cudaMalloc((void**)&d_maxminusdyminus,N*sizeof(double));
	cudaMalloc((void**)&d_mindyplus,N*sizeof(double));
	cudaMalloc((void**)&d_minminusdyminus,N*sizeof(double));


	cudaMalloc((void**)&d_nplusx,N*sizeof(double));
	cudaMalloc((void**)&d_nplusy,N*sizeof(double));
	cudaMalloc((void**)&d_nminusx,N*sizeof(double));
	cudaMalloc((void**)&d_nminusy,N*sizeof(double));
	cudaMalloc((void**)&d_gradphimax,N*sizeof(double));
	cudaMalloc((void**)&d_gradphimin,N*sizeof(double));
	cudaMalloc((void**)&d_curvature,N*sizeof(double));
	cudaMalloc((void**)&d_F,N*sizeof(double));
	cudaMalloc((void**)&d_gradphi,N*sizeof(double));
	cudaMalloc((void**)&d_phi,N*sizeof(double));
	cudaMalloc((void**)&d_phi_new,N*sizeof(double));
	cudaMalloc((void**)&d_D,N*sizeof(double));
}

void Free_Memory_GPU()
{
	//Free Memory
	cudaFree(d_dx);
	cudaFree(d_dxplus);
	cudaFree(d_dxminus);
	cudaFree(d_dxplusy);
	cudaFree(d_dxminusy);
	cudaFree(d_maxdxplus);
	cudaFree(d_maxminusdxminus);
	cudaFree(d_mindxplus);
	cudaFree(d_minminusdxminus);

	cudaFree(d_dy);
	cudaFree(d_dyplus);
```

```
	cudaFree(d_dyminus);
	cudaFree(d_dyplusx);
	cudaFree(d_dyminusx);
	cudaFree(d_maxdyplus);
	cudaFree(d_maxminusdyminus);
	cudaFree(d_mindyplus);
	cudaFree(d_minminusdyminus);

	cudaFree(d_nplusx);
	cudaFree(d_nplusx);
	cudaFree(d_nminusx);
	cudaFree(d_nminusy);
	cudaFree(d_gradphimax);
	cudaFree(d_gradphimin);
	cudaFree(d_curvature);
	cudaFree(d_F);
	cudaFree(d_D);
	cudaFree(d_gradphi);
	cudaFree(d_phi);
	cudaFree(d_phi_new);

	free(phi);
	free(D);
}

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

}

void Alokasi_Memori_CPU()
{
	phi = (double *)malloc(imageH*imageW*sizeof(double));
	D = (double *)malloc(imageH*imageW*sizeof(double));

	//========DX=======//
```

```
	h_dx = (double *)malloc(imageH*imageW*sizeof(double));
	h_dxplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_dxminus = (double *)malloc(imageH*imageW*sizeof(double));
	h_dxplusy = (double *)malloc(imageH*imageW*sizeof(double));
	h_dxminusy = (double *)malloc(imageH*imageW*sizeof(double));
	h_maxdxplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_maxminusdxminus = (double *)malloc(imageH*imageW*sizeof(double));
	h_mindxplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_minminusdxminus = (double *)malloc(imageH*imageW*sizeof(double));

	//========DY=======//
	h_dy = (double *)malloc(imageH*imageW*sizeof(double));
	h_dyplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_dyminus = (double *)malloc(imageH*imageW*sizeof(double));
	h_dyplusx = (double *)malloc(imageH*imageW*sizeof(double));
	h_dyminusx = (double *)malloc(imageH*imageW*sizeof(double));
	h_maxdyplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_maxminusdyminus = (double *)malloc(imageH*imageW*sizeof(double));
	h_mindyplus = (double *)malloc(imageH*imageW*sizeof(double));
	h_minminusdyminus = (double *)malloc(imageH*imageW*sizeof(double));

	//====Gradien Phi=====//
	h_gradphimax = (double *)malloc(imageH*imageW*sizeof(double));
	h_gradphimin = (double *)malloc(imageH*imageW*sizeof(double));
	h_gradphi = (double *)malloc(imageH*imageW*sizeof(double));

	//====n=======//
	h_nplusx = (double *)malloc(imageH*imageW*sizeof(double));
	h_nplusy = (double *)malloc(imageH*imageW*sizeof(double));
	h_nminusx = (double *)malloc(imageH*imageW*sizeof(double));
	h_nminusy = (double *)malloc(imageH*imageW*sizeof(double));

	h_curvature = (double *)malloc(imageH*imageW*sizeof(double));
	h_F = (double *)malloc(imageH*imageW*sizeof(double));
}

void Free_Memory_CPU()
{
	//=======DX======//
	free(h_dx);
	free(h_dxplus);
	free(h_dxminus);
	free(h_dxminusy);
	free(h_dxplusy);
	free(h_maxdxplus);
	free(h_maxminusdxminus);
	free(h_mindxplus);
	free(h_minminusdxminus);

	//=======DX======//
	free(h_dy);
	free(h_dyplus);
	free(h_dyminus);
```

```
	free(h_dyplusx);
	free(h_dyminusx);
	free(h_maxdyplus);
	free(h_maxminusdyminus);
	free(h_mindyplus);
	free(h_minminusdyminus);

	//====Gradien Phi=====//
	free(h_gradphi);
	free(h_gradphimax);
	free(h_gradphimin);

	//====n=======//
	free(h_nplusx);
	free(h_nplusy);
	free(h_nminusx);
	free(h_nminusy);

	free(h_F);
	free(h_curvature);
	free(D);
	free(phi);
}

void update_phi()
{
	//=========================DX=======================
	//dx
	double phi_ip1 = 0.0;
	double phi_im1 = 0.0;

	for (int j = 0; j < imageH; j++)
	{
		for (int i = 0; i < imageW; i++)
		{
			int i1 = j*imageW+i;

			if(i == 0)
			{
				phi_im1 = 0;
				phi_ip1 = phi[i1 + 1];
			}
			else if(i==imageW-1)
			{
				phi_ip1=0;
				phi_im1 = phi[i1 - 1];
			}

			else //if(i!=0 || i!= imageH-1)
			{
				phi_im1 = phi[i1 - 1];
				phi_ip1 = phi[i1 + 1];
			}
```

```
                h_dx[i1] = (phi_ip1 - phi_im1)/2;

        }
}

//dxplus
phi_ip1 = 0.0;

for (int j = 0; j < imageH; j++)
{
        for (int i = 0; i < imageW; i++)
        {
                int i1 = j*imageW+i;

                if(i ==  imageW-1)
                {
                        phi_ip1=0.0;
                }
                else
                {
                        phi_ip1 = phi[i1 + 1];
                }

                h_dxplus[i1] = phi_ip1 - phi[i1];

        }
}

//dxminus
phi_im1 = 0.0;

for (int j = 0; j < imageH; j++)
{
        for (int i = 0; i < imageW; i++)
        {
                int i1 = j*imageW+i;

                if(i==0)
                {
                        phi_im1=0;
                }

                else
                {
                        phi_im1 = phi[i1 - 1];
                }

                h_dxminus[i1] = phi[i1] - phi_im1;

        }
}
```

```
//dxplusy
double phi_ip1_jp1 = 0.0;
double phi_im1_jp1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(j == imageH - 1)
		{
			phi_ip1_jp1 = 0.0;
			phi_im1_jp1 = 0.0;
		}

		else if(i == 0)
		{
			phi_ip1_jp1 = phi[i1 + 1 + imageW];
			phi_im1_jp1 = 0.0;
		}

		else if(i == imageW - 1)
		{
			phi_ip1_jp1 = 0.0;
			phi_im1_jp1 = phi[i1 - 1 + imageW];
		}

		else
		{
			phi_ip1_jp1 = phi[i1 + 1 + imageW];
			phi_im1_jp1 = phi[i1 - 1 + imageW];
		}

		h_dxplusy[i1] = (phi_ip1_jp1 - phi_im1_jp1)/2;
	}
}

//dxminusy
double phi_ip1_jm1 = 0.0;
double phi_im1_jm1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(j == 0)
		{
			phi_ip1_jm1 = 0.0;
```

```
                        phi_im1_jm1 = 0.0;
                }

                else if (i == 0)
                {
                        phi_ip1_jm1 = phi[i1 + 1 - imageW];
                        phi_im1_jm1 = 0.0;
                }

                else if (i == imageW - 1)
                {
                        phi_ip1_jm1 = 0.0;
                        phi_im1_jm1 = phi[i1 - 1 - imageW];
                }

                else
                {
                        phi_ip1_jm1 = phi[i1 + 1 - imageW];
                        phi_im1_jm1 = phi[i1 - 1 - imageW];
                }

                h_dxminusy[i1] = (phi_ip1_jm1 - phi_im1_jm1)/2;
        }
}


//maxdxplus
for (int i = 0; i < N; i++)
{
        h_maxdxplus[i] = h_dxplus[i];
}

for (int i = 0; i < N; i++)
{
        if(h_maxdxplus[i] < 0)
        {
                h_maxdxplus[i] = 0;
        }
        else
        {
                h_maxdxplus[i] *= h_maxdxplus[i];
        }
}


//maxminusdxminus
for (int i = 0; i < N; i++)
{
        h_maxminusdxminus[i] = -h_dxminus[i];
}

for (int i = 0; i < N; i++)
{
```

```
        if(h_maxminusdxminus[i] < 0)
        {
                h_maxminusdxminus[i] = 0;
        }
        else
        {
                h_maxminusdxminus[i] *= h_maxminusdxminus[i];
        }
}


//mindxplus
for (int i = 0; i < N; i++)
{
        h_mindxplus[i] = h_dxplus[i];
}

for (int i = 0; i < N; i++)
{
        if(h_mindxplus[i] > 0)
        {
                h_mindxplus[i] = 0;
        }
        else
        {
                h_mindxplus[i] *= h_mindxplus[i];
        }
}


//minminusdxminus
for (int i = 0; i < N; i++)
{
        h_minminusdxminus[i] = -h_dxminus[i];
}

for (int i = 0; i < N; i++)
{
        if(h_minminusdxminus[i] > 0)
        {
                h_minminusdxminus[i] = 0;
        }
        else
        {
                h_minminusdxminus[i] *= h_minminusdxminus[i];
        }
}


//================DY============================//

//dy
double phi_jp1 = 0.0;
```

```
double phi_jm1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(j == 0)
		{
			phi_jm1 = 0.0;
			phi_jp1 = phi[i1 + imageW];
		}
		else if(j == imageH - 1)
		{
			phi_jp1 = 0.0;
			phi_jm1 = phi[i1 - imageW];
		}

		else
		{
			phi_jm1 = phi[i1 - imageW];
			phi_jp1 = phi[i1 + imageW];
		}

		h_dy[i1] = (phi_jp1 - phi_jm1)/2;

	}
}


//dyplus
phi_jp1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(j == imageH - 1)
		{
			phi_jp1 = 0;
		}
		else
		{
			phi_jp1 = phi[i1 + imageW];
		}

		h_dyplus[i1] = phi_jp1 - phi[i1];

	}
}
```

```
//dyminus
phi_jm1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(j == 0)
		{
			phi_jm1 = 0.0;
		}

		else
		{
			phi_jm1 = phi[i1 - imageW];
		}

		h_dyminus[i1] = phi[i1] - phi_jm1;

	}
}


//dyplusx
phi_ip1_jp1 = 0.0;
phi_ip1_jm1 = 0.0;

for (int j = 0; j < imageH; j++)
{
	for (int i = 0; i < imageW; i++)
	{
		int i1 = j*imageW+i;

		if(i == imageW - 1)
		{
			phi_ip1_jp1 = 0.0;
			phi_ip1_jm1 = 0.0;
		}

		else if (j == 0)
		{
			phi_ip1_jp1 = phi[i1 + 1 + imageW];
			phi_ip1_jm1 = 0.0;
		}

		else if (j == imageH - 1)
		{
			phi_ip1_jp1 = 0.0;
			phi_ip1_jm1 = phi[i1 + 1 - imageW];
```

```
                }

                else
                {
                        phi_ip1_jp1 = phi[i1 + 1 + imageW];
                        phi_ip1_jm1 = phi[i1 + 1 - imageW];
                }

                h_dyplusx[i1] = (phi_ip1_jp1 - phi_ip1_jm1)/2;
        }
}


//dyminusx
phi_im1_jp1 = 0.0;
phi_im1_jm1 = 0.0;

for (int j = 0; j < imageH; j++)
{
        for (int i = 0; i < imageW; i++)
        {
                int i1 = j*imageW+i;

                if(i == 0)
                {
                        phi_im1_jp1 = 0.0;
                        phi_im1_jm1 = 0.0;
                }

                else if(j == 0)
                {
                        phi_im1_jp1 = phi[i1 - 1 + imageW];
                        phi_im1_jm1 = 0.0;
                }

                else if(j == imageH - 1)
                {
                        phi_im1_jp1 = 0.0;
                        phi_im1_jm1 = phi[i1 - 1 - imageW];
                }

                else
                {
                        phi_im1_jp1 = phi[i1 - 1 + imageW];
                        phi_im1_jm1 = phi[i1 - 1 - imageW];
                }

                h_dyminusx[i1] = (phi_im1_jp1 - phi_im1_jm1)/2;
        }
}


//maxdyplus
```

```
for (int i = 0; i < N; i++)
{
	h_maxdyplus[i] = h_dyplus[i];
}

for (int i = 0; i < N; i++)
{
	if(h_maxdyplus[i] < 0)
	{
		h_maxdyplus[i] = 0;
	}
	else
	{
		h_maxdyplus[i] *= h_maxdyplus[i];
	}
}


//maxminusdyminus
for (int i = 0; i < N; i++)
{
	h_maxminusdyminus[i] = -h_dyminus[i];
}

for (int i = 0; i < N; i++)
{
	if(h_maxminusdyminus[i] < 0)
	{
		h_maxminusdyminus[i] = 0;
	}
	else
	{
		h_maxminusdyminus[i] *= h_maxminusdyminus[i];
	}
}


//mindyplus
for (int i = 0; i < N; i++)
{
	h_mindyplus[i] = h_dyplus[i];
}

for (int i = 0; i < N; i++)
{
	if(h_mindyplus[i] > 0)
	{
		h_mindyplus[i] = 0;
	}
	else
	{
		h_mindyplus[i] *= h_mindyplus[i];
	}
```

```
	}


	//minminusdyminus
	for (int i = 0; i < N; i++)
	{
		h_minminusdyminus[i] = -h_dyminus[i];
	}

	for (int i = 0; i < N; i++)
	{
		if(h_minminusdyminus[i] > 0)
		{
			h_minminusdyminus[i] = 0;
		}
		else
		{
			h_minminusdyminus[i] *= h_minminusdyminus[i];
		}
	}


	//====Gradien Phi=====//
	//gradphimax
	for (int i = 0; i < N; i++)
	{
		h_gradphimax[i] =
sqrt((sqrt(h_maxdxplus[i]+h_maxminusdxminus[i]))*(sqrt(h_maxdxplus[i]+h_maxminu
sdxminus[i])) +
(sqrt(h_maxdyplus[i]+h_maxminusdyminus[i]))*(sqrt(h_maxdyplus[i]+h_maxminusdymi
nus[i])));
	}

	//gradphimin
	for (int i = 0; i < N; i++)
	{
		h_gradphimin[i] =
sqrt((sqrt(h_mindxplus[i]+h_minminusdxminus[i]))*(sqrt(h_mindxplus[i]+h_minminu
sdxminus[i])) +
(sqrt(h_mindyplus[i]+h_minminusdyminus[i]))*(sqrt(h_mindyplus[i]+h_minminusdymi
nus[i])));
	}


	//====n=======//
	//nplusx
	for (int i = 0; i < N; i++)
	{
		h_nplusx[i] = h_dxplus[i]/sqrt(FLT_EPSILON +
(h_dxplus[i]*h_dxplus[i]) +
((h_dyplusx[i]+h_dy[i])*(h_dyplusx[i]+h_dy[i])*0.25));
	}
```

```
	//nplusy
	for (int i = 0; i < N; i++)
	{
		h_nplusy[i] = h_dyplus[i]/sqrt(FLT_EPSILON +
(h_dyplus[i]*h_dyplus[i]) +
((h_dxplusy[i]+h_dx[i])*(h_dxplusy[i]+h_dx[i])*0.25));
	}

	//nminusx
	for (int i = 0; i < N; i++)
	{
		h_nminusx[i] = h_dxminus[i]/sqrt(FLT_EPSILON +
(h_dxminus[i]*h_dxminus[i]) +
((h_dyminusx[i]+h_dy[i])*(h_dyminusx[i]+h_dy[i])*0.25));
	}

	//nminusy
	for (int i = 0; i < N; i++)
	{
		h_nminusy[i] = h_dyminus[i]/sqrt(FLT_EPSILON +
(h_dyminus[i]*h_dyminus[i]) +
((h_dxminusy[i]+h_dx[i])*(h_dxminusy[i]+h_dx[i])*0.25));
	}
	//=============================================//


	//curvature
	for (int i = 0; i < N; i++)
	{
		h_curvature[i] = ((h_nplusx[i]-h_nminusx[i])+(h_nplusy[i]-
h_nminusy[i]))/2;
	}


	//F
	for (int i = 0; i < N; i++)
	{
		h_F[i] = -(ALPHA * D[i]) + ((1 - ALPHA) * h_curvature[i]);
	}


	//gradphi
	for (int i = 0; i < N; i++)
	{
		if(h_F[i] > 0)
		{
			h_gradphi[i] = h_gradphimax[i];
		}
		else
		{
			h_gradphi[i] = h_gradphimin[i];
		}
	}
```

```
	//PHI
	for (int i = 0; i < N; i++)
	{
		phi[i] = phi[i] + DT * h_F[i] * h_gradphi[i];
	}

}

void Image_Segmentation_CPU(int iterasi)
{
	Mat  ambil, clone_temp;
	Mat temp, image_phi;

	//Proses Akses setiap frame yang telah disimpan pada variabel
Image_Frames
	for (vector<Mat>::iterator iter = Images_Frame.begin(); iter !=
Images_Frame.end(); iter++)
	{
		//Proses penyelinan frame dari Image_Frame ke variabel temp
		temp = *iter;

		clone_temp = temp.clone();

		//Inisialisasi Nilai dari D
		InisialisaiD(clone_temp);

		//Inisialisasi nilai dari Phi
		init_phi(clone_temp, inisialisasi);

		//Melakukan proses segmentasi citra pada setiap frame
		for (int its = 0; its < iterasi; its++)
		{
			//melakukan update nilai phi
			update_phi();
		}

		image_phi = clone_temp;

		//Konversi citra 8UC1 ke 64FC1
		image_phi.convertTo(image_phi, CV_64FC1);

		//Proses mengubah nilai phi yang telah diolah menjadi citra (1D
menjadi 2D)
		for (int y = 0; y < imageW; y++)
		{
			for (int x = 0; x < imageH; x++)
			{
				int i1 = y + x * imageW;

				image_phi.at<double>(x,y) = phi[i1];
			}
		}
```

```
                //Konversi citra 64FC1 ke 8UC1
                image_phi.convertTo(image_phi, CV_8UC1);

                ambil = clone_temp;
                ambil_citra(clone_temp, &ambil, image_phi, imageH, imageW);

                //Memasukkan frame yang telah diolah ke variabel Phi_Frame_CPU
                Phi_Frame_CPU.push_back(ambil);
        }
}

void WriteVideoCPU(double TotalFrame, int width, int height, double FPS)
{
        Mat tempframe, frame_hasil;

        // Open a video file for writing (the MP4V codec works on OS X and
Windows)
        //VideoWriter out("output.mov", CV_FOURCC('m','p', '4', 'v'), FPS, Size
(width, height));
        VideoWriter out("output_cpu.mov", CV_FOURCC('m','p', '4', 'v'), FPS,
Size(width, height));
     if(!out.isOpened()) {
          printf("Error! Unable to open video file for output.");
          exit(-1);
     }

         for (vector<Mat>::iterator iter = Phi_Frame_CPU.begin(); iter !=
Phi_Frame_CPU.end(); iter++)
        {
                tempframe = *iter;
                cvtColor(tempframe,frame_hasil,CV_GRAY2RGB);
                out << frame_hasil;
        }


}

void SegmentasiVideo_LevelSet_CPU()
{
        //====Variabel Waktu CPU====//
        clock_t start_cpu, stop_cpu;
    double cpu_time_used;

        //Alokasi Memori CPU
        Alokasi_Memori_CPU();

        //timer dimulai
        start_cpu = clock();
        //Proses segemntasi video
        Image_Segmentation_CPU(iterasi);
        //timer berhenti
        stop_cpu = clock();
```

```
	//menghitung selisih waktu selesai dan waktu mulai proses segmentasi
video
	cpu_time_used = ((double)(stop_cpu - start_cpu))/CLOCKS_PER_SEC;

	save_to_file(phi, "data_phi_cpu.txt");

	printf("Waktu Segmentasi Video = %lf detik\n", cpu_time_used);

	//membersihkan memori yang telah digunakan
	Free_Memory_CPU();

	printf("\nProses Segmentasi pada CPU Selesai\n");
}

void update_phi_GPU()
{
	//Inisialisai jumlah blok dan grid yang digunakan
	dim3 block(32/4,32/4);
	dim3 grid((imageW+block.x-1)/block.x, (imageH+block.y-1)/block.y);

	dx<<<grid,block>>>(d_phi,d_dx,imageH,imageW);

	dxplus<<<grid,block>>>(d_phi,d_dxplus,imageH,imageW);
	dxminus<<<grid,block>>>(d_phi,d_dxminus,imageH,imageW);
	dxplusy<<<grid,block>>>(d_phi,d_dxplusy,imageH,imageW);
	dxminusy<<<grid,block>>>(d_phi,d_dxminusy,imageH,imageW);
	maxdxplus<<<grid,block>>>(d_dxplus,d_maxdxplus,imageH,imageW);
	maxminusdxminus<<<grid,block>>>(d_dxminus,d_maxminusdxminus,imageH,image
W);
	mindxplus<<<grid,block>>>(d_dxplus,d_mindxplus,imageH,imageW);
	minminusdxminus<<<grid,block>>>(d_dxminus,d_minminusdxminus,imageH,image
W);

	dy<<<grid,block>>>(d_phi,d_dy,imageH,imageW);
	dyplus<<<grid,block>>>(d_phi,d_dyplus,imageH,imageW);
	dyminus<<<grid,block>>>(d_phi,d_dyminus,imageH,imageW);
	dyplusx<<<grid,block>>>(d_phi,d_dyplusx,imageH,imageW);
	dyminusx<<<grid,block>>>(d_phi,d_dyminusx,imageH,imageW);
	maxdyplus<<<grid,block>>>(d_dyplus,d_maxdyplus,imageH,imageW);
	maxminusdyminus<<<grid,block>>>(d_dyminus,d_maxminusdyminus,imageH,image
W);
	mindyplus<<<grid,block>>>(d_dyplus,d_mindyplus,imageH,imageW);
	minminusdyminus<<<grid,block>>>(d_dyminus,d_minminusdyminus,imageH,image
W);

	gradphimax<<<grid,block>>>(d_gradphimax,d_maxdxplus,d_maxminusdxminus,d_
maxdyplus, d_maxminusdyminus, imageH,imageW);
	gradphimin<<<grid,block>>>(d_gradphimin,d_mindxplus,d_minminusdxminus,d_
mindyplus, d_minminusdyminus, imageH,imageW);

	nplusx<<<grid,block>>>(d_nplusx,d_dxplus,d_dyplusx,d_dy,FLT_EPSILON,
imageH,imageW);
```

```
	nplusy<<<grid,block>>>(d_nplusy,d_dyplus,d_dxplusy,d_dx, FLT_EPSILON,
imageH,imageW);
	nminusx<<<grid,block>>>(d_nminusx,d_dxminus,d_dyminusx,d_dy,
FLT_EPSILON, imageH,imageW);
	nminusy<<<grid,block>>>(d_nminusy,d_dyminus,d_dxminusy,d_dx,
FLT_EPSILON, imageH,imageW);

	curvature<<<grid,block>>>(d_curvature,d_nplusx,d_nminusx,d_nplusy,
d_nminusy, imageH,imageW);
	F<<<grid,block>>>(d_F,d_D,d_curvature, ALPHA, imageH,imageW);
	gradphi<<<grid,block>>>(d_gradphi,d_F,d_gradphimax,d_gradphimin,
imageH,imageW);

	update_phi<<<grid,block>>>(d_phi_new,d_phi,d_F, d_gradphi, DT,
imageH,imageW);

	copy_data<<<grid,block>>>(d_phi,d_phi_new, imageH,imageW);


}

void Image_Segmentation_GPU(int iterasi)
{
	Mat temp, image_phi;
	Mat clone_temp, ambil;

	//Proses Akses setiap frame yang telah disimpan pada variabel
Image_Frames
	for (vector<Mat>::iterator iter = Images_Frame.begin(); iter !=
Images_Frame.end(); iter++)
	{
		//Proses penyelinan frame dari Image_Frame ke variabel temp
		temp = *iter;

		clone_temp = temp.clone();

		//Inisialisasi Nilai dari D
		InisialisaiD(clone_temp);

		//Inisialisasi nilai dari Phi
		init_phi(clone_temp, inisialisasi);

		//Menyalin memori dari hosy(CPU) ke device(GPU)
		cudaMemcpy(d_phi, phi, N*sizeof(double), cudaMemcpyHostToDevice);
		cudaMemcpy(d_D, D, N*sizeof(double), cudaMemcpyHostToDevice);

		//Melakukan proses segmentasi citra pada setiap frame
		for (int its = 0; its < iterasi; its++)
		{
			//melakukan update nilai phi
			update_phi_GPU();
		}
```

```
            //Menyalin Hasil GPU ke CPU
            cudaMemcpy(phi, d_phi,N*sizeof(double), cudaMemcpyDeviceToHost);

            image_phi = clone_temp;

            //Konversi citra 8UC1 ke 64FC1
            image_phi.convertTo(image_phi, CV_64FC1);

            //Proses mengubah nilai phi yang telah diolah menjadi citra (1D
menjadi 2D)
            for (int y = 0; y < imageW; y++)
            {
                  for (int x = 0; x < imageH; x++)
                  {
                        int i1 = y + x * imageW;

                        image_phi.at<double>(x,y) = phi[i1];
                  }
            }

            //Konversi citra 64FC1 ke 8UC1
            image_phi.convertTo(image_phi, CV_8UC1);

            ambil = clone_temp;
            ambil_citra(clone_temp, &ambil, image_phi,imageH, imageW);

            //Menyimpan frame yang telah diolah ke variabel Phi_Frame_GPU
            Phi_Frame_GPU.push_back(ambil);

      }

}

void SegmentasiVideo_LevelSet_GPU()
{
      //====Variabel Waktu GPU====//
      cudaEvent_t start,stop;
      cudaEventCreate(&start);
      cudaEventCreate(&stop);
      float elapsedTime;


      //Alokasi Memori GPU
      Alokasi_Memori_GPU();

      //Komputasi Pada GPU
      printf("Proses Komputasi Pada GPU .....\n");

      //timer dimulai
      cudaEventRecord(start,0);

      Image_Segmentation_GPU(iterasi);
```

```
        cudaDeviceSynchronize();

        //timer berhenti
        cudaEventRecord(stop,0);
        cudaEventSynchronize(stop);

        //menghitung selisih waktu selesai dan waktu mulai proses segmentasi
video
        cudaEventElapsedTime(&elapsedTime,start,stop);
        printf("\nElapsed Time \t:%f detik\n",elapsedTime/1000);

        save_to_file(phi,"data_phi_video_gpu.txt");

        cudaEventDestroy(start);
        cudaEventDestroy(stop);

        //membersihkan memori yang telah digunakan
        Free_Memory_GPU();

        cudaDeviceReset();

        printf("\nProses Segmentasi pada GPU Selesai\n");
}
```