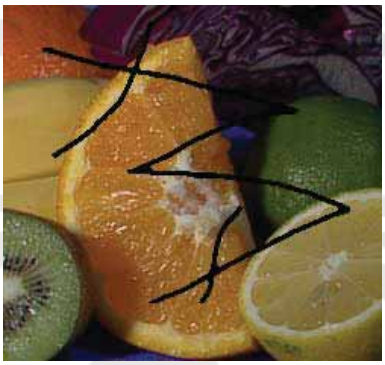
BAB III
METODOLOGI PENELITIAN

III.1. Bahan Penelitian

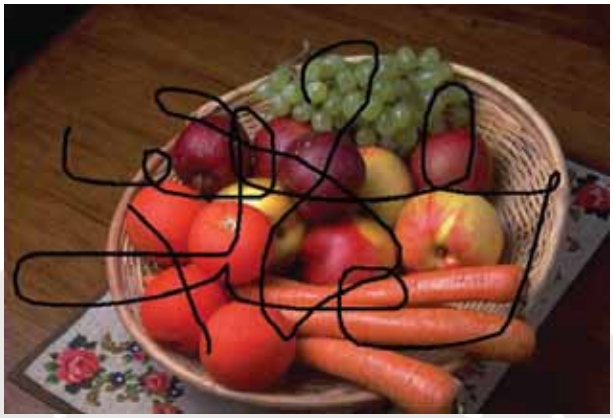
Pada penelitian ini terdapat beberapa bahan materi (dataset) pada citra uji, citra yang digunakan sebagai bahan uji terdiri dari empat citra berwarna dan empat citra *grayscale*. Masing-masing citra yang digunakan meiliki ukuran dan kerusakan yang berbeda-beda, data citra adalah sebagai berikut :

1. Citra fruits.bmp
   Citra fruits.bmp adalah citra berwarna dengan ukuran 250 x 234 piksel. Citra ini digolongkan kedalam kategori small(S) dalam pengujian.



Gambar 3.1. fruits.bmp

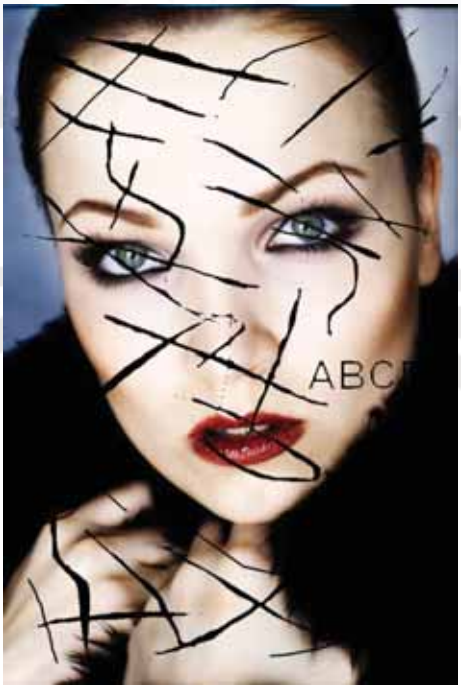
2. Citra fruit.bmp
   Citra fruit.bmp adalah citra berwarna dengan ukuran 600 x 406 piksel. Citra ini digolongkan kedalam kategori medium(M) dalam pengujian.



Gambar 3.2. fruit.bmp

3. Citra tarja.bmp

   Citra tarja.bmp adalah citra berwarna dengan ukuran 682 x 1024 piksel. Citra ini digolongkan kedalam kategori large(L) dalam pengujian.



Gambar 3.3. tarja.bmp

4. Citra bromo.bmp

   Citra bromo.bmp adalah citra berwarna dengan ukuran 2048 x 1536 piksel. Citra ini digolongkan kedalam kategori extra large(XL) dalam pengujian.

Gambar 3.4. bromo.bmp

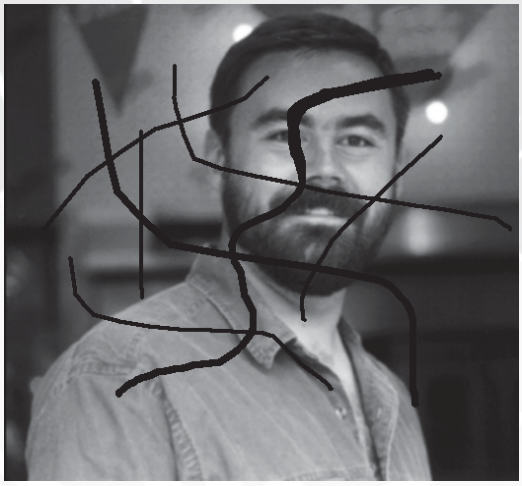
5. Citra lena.bmp

   Citra lena.bmp adalah citra *gray* dengan ukuran 250 x 250 piksel. Citra ini digolongkan kedalam kategori small(S) dalam pengujian.

Gambar 3.5. lena.bmp

6. Citra portrait.bmp

   Citra portrait.bmp adalah citra *gray* dengan ukuran 522 x 486 piksel. Citra ini digolongkan kedalam kategori medium(M) dalam pengujian.



Gambar 3.6. portrait.bmp

7. Citra malioboro.bmp

   Citra malioboro.bmp adalah citra *gray* dengan ukuran 1024 x 512 piksel. Citra ini digolongkan kedalam kategori large(L) dalam pengujian.

Gambar 3.7. malioboro.bmp

8. Citra beach.bmp

   Citra beach.bmp adalah citra *gray* dengan ukuran 2048 x 1536 piksel. Citra ini digolongkan kedalam kategori extra large(XL) dalam pengujian.

Gambar 3.8. beach.bmp

III.2. Alat Penelitian

Adapun alat penelitian yang mendukung penelitian ini terdiri dari 2 komponen yaitu *hardware* dan *software*. Berikut adalah spesifikasi *hardware* dan *software* yang digunakan :

1. *Software*

   Perangkat lunak yang dibutuhkan untuk mengoperasikan program *Perona-Malik Inpainting* adalah sebagai berikut :

a. Nama : C/C++ Microsoft VS 2010
   Sumber : Microsoft
   Sebagai aplikasi bahasa pemrograman untuk membuat project program *Perona-Malik Inpainting*.

b. Nama : Windows 7
   Sumber : Microsoft
   Sebagai sistem operasi dimana *Perona-malik Inpainting* berjalan.

c. Nama : CUDA
   Sumber : NVIDIA
   Sebagai *library* komputasi paralel yang diterapkan dalam project.

d. Nama : OpenCV
   Sumber : Itseez
   Sebagai *library* untuk pemrosesan citra digital yang diterapkan dalam project.

2. Hardware

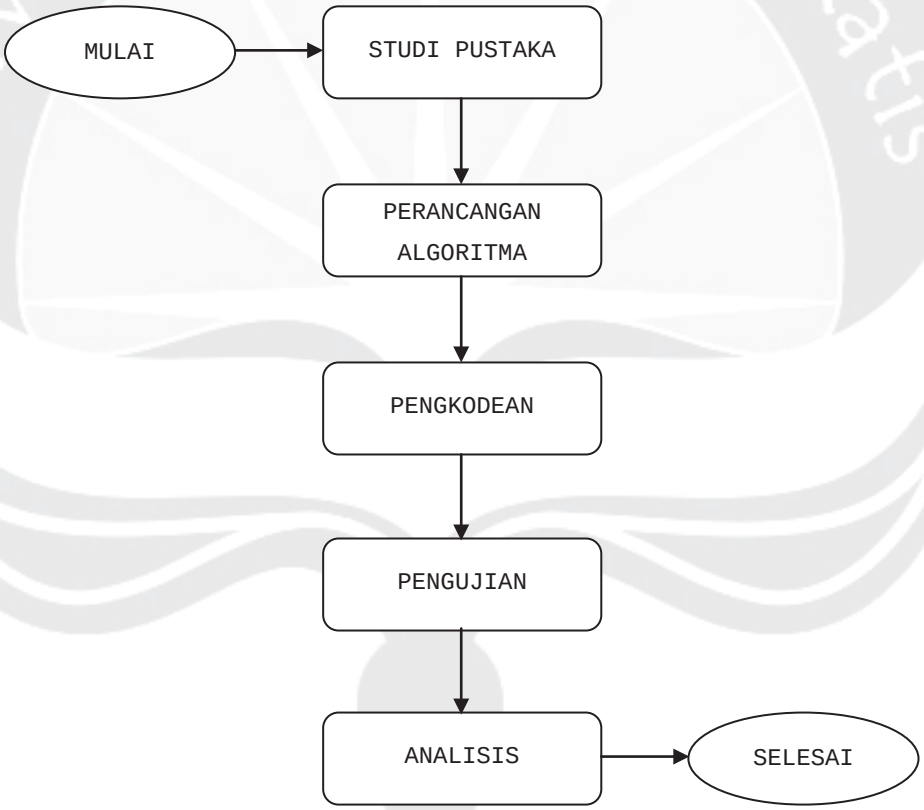
Perangkat keras yang digunakan untuk mendukung penelitian program *Perona-Malik Inpainting* adalah sebagai berikut :

a. *Personal Computer* dengan spesifikasi CPU
   CPU Name : Intel® Core™ i7-3770K CPU @3.5Ghz
   CPU Memory : 16384 MB

b. *Personal Computer* dengan spesifikasi GPU
   GPU Name : NVIDIA GeForce GTX 670
   GPU Memory : 2021 MB

III.3. Langkah-langkah Penelitian

Adapun beberapa tahap yang akan dilakukan penulis untuk melakukan penelitian ini, seperti: studi pustaka, perancangan algoritma, pengkodean, pengujian, dan analisis pada Gambar 3.9 berikut ini :



Gambar 3.9. *flowchart* langkah-langkah penelitian

III.3.1. Studi Pustaka

Studi Pustaka merupakan langkah pertama pada proses penyelesaian masalah pada penelitian ini. Studi Pustaka ini berguna untuk memenuhi informasi yang dibutuhkan dalam menyelesaikan penelitian ini. Studi pustaka dilakukan dengan membaca dan mempelajari beberapa referensi dari beberapa buku dan jurnal yang berhubungan dengan inpainting, metode Perona-Malik, komputasi paralel, dan GPU CUDA serta mempelajari beberapa hipotesa dari penelitian-penelitian sebelumnya.

III.3.2. Perancangan Algoritma

Algoritma yang digunakan dalam melakukan Inpainting pada citra digital pada penelitian ini adalah algoritma *inpainting*. Algoritma *inpainting* akan dikombinasikan dengan dengan algoritma *Perona-Malik*. Algoritma ini akan diimplementasikan pada CPU secara Serial dan pada GPU secara Paralel.

III.3.2.1. Serial Perona-Malik Inpainting

Algoritma Perona-Malik Inpainting pada CPU dapat dinyatakan sebagai *pseudocode* berikut:

1. Inisialisasi semua parameter d_in, d_out, M, N, lambda, G dengan citra dan *masking* yang diinputkan.

2. Deklarasi semua variable tambahan delta, $c_{N,S,E,W}$, dt, K dengan nilai awal dt = 0.1 dan nilai awal K = 10.

3. **For** iterasi = 1 **to** iterasi maksimum **do**:

    a. **For each** dimensi y **do**:

        i. **For each** dimensi x **do**:

            1. Cek index terendah dan index tertinggi berdasarkan dimensi y dan dimensi x

            2. Tentukan index utama dan index disekitar index utama

            3. Simpan nilai index disekitar pada variable left, right, top, bottom

            4. Tentukan nilai delta

                a. deltaN ← d_in[left]-d_in[index]

                b. deltaS ← d_in[right]-d_in[index]

                c. deltaE ← d_in[top]-d_in[index]

                d. deltaW ← d_in[bottom]-d_in[index]

            5. pilih metode

                a. **if** metode == 1:

                    i. $c_{N,S,E,W}$ ← PM1(non linier)

                b. **if** metode == 2:

i. $c_{N,S,E,W}$ ← PM2(non linier)

c. **if** metode == 3:

i. $c_{N,S,E,W}$ ← heat(linier)

6. proses inpainting dengan meng-*update* setiap nilai d_out dengan hasil perhitungan disetiap pikselnya

4. *Update* nilai d_in dengan nilai d_out yang baru disetiap iterasi

III.3.2.2. Paralel Perona-Malik Inpainting

Algoritma Perona-Malik Inpainting pada GPU dapat dinyatakan sebagai *pseudocode* berikut:

1. Inisialisasi semua parameter d_in, d_out, M, N, lambda, G dengan citra dan *masking* yang diinputkan.
2. Deklarasi semua variable tambahan delta, $c_{N,S,E,W}$, dt, K dengan nilai awal dt = 0.1 dan nilai awal K = 10.
3. *Copy* semua variable dan parameter dari host ke device
4. **For** iterasi = 1 **to** iterasi maksimum **do**:
    a. Tentukan dimensi y dan x secara paralel berdasarkan *thread, block,* dan *grid*

b. Cek index terendah dan index tertinggi berdasarkan dimensi y dan dimensi x secara paralel

c. Tentukan index utama dan index disekitar index utama secara paralel

d. Simpan nilai index disekitar pada variable left, right, top, bottom

e. Tentukan nilai delta

   i. deltaN ← d_in[left]-d_in[index]

   ii. deltaS ← d_in[right]-d_in[index]

   iii. deltaE ← d_in[top]-d_in[index]

   iv. deltaW ← d_in[bottom]-d_in[index]

f. pilih metode

   i. **if** metode == 1:

      1. $c_{N,S,E,W}$ ← PM1(non linier)

   ii. **if** metode == 2:

      1. $c_{N,S,E,W}$ ← PM2(non linier)

   iii. **if** metode == 3:

      1. $c_{N,S,E,W}$ ← heat(linier)

g. proses inpainting dengan meng-*update* setiap nilai d_out dengan hasil perhitungan disetiap pikselnya secara paralel.

5. *Update* nilai d_in dengan nilai d_out yang baru disetiap iterasi secara paralel
6. *Copy* nilai d_out dari device ke host

III.3.3. Pengkodean

Implementasi algoritma dibagi menjadi 2 bagian, antara lain implementasi pada CPU dan GPU, dimana algoritma yang diimplementasikan pada masing-masing prosesor adalah algoritma Heat, Perona-Malik 1, dan Perona-Malik 2. Secara garis besar, proses dan algoritma *inpainting* yang dilakukan pada CPU dan GPU adalah hampir sama, yang membedakan hanyalah prosesor dan memori yang digunakan saat proses *inpainting* berlangsung.

III.3.3.1. Implementasi Algoritma Secara Umum

Sebelum masuk pada implementasi algoritma *Perona-Malik*, akan dijelaskan terlebih dahulu komponen-komponen *library*, variable, dan prosedur apa saja yang digunakan untuk melakukan proses *inpainting* ini.

```
//--Library OpenCV2.3
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

//--library CUDA
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

//--library C++
#include <stdio.h>
#include <conio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <windows.h>
#include <ctype.h>

#define I2D(nj,i,j)(((nj)*(i))+j)
```

Kode 3.1. Library

Terdapat tiga jenis *library* yang digunakan dalam program ini. Pertama adalah *library* OpenCV, OpenCV yang digunakan adalah versi 2.3.0. *Library* ini disertakan untuk mengelola citra yang akan digunakan dalam *inpainting*. Mengelola maksudnya adalah membaca(*load*), menulis(*save*), dan mendapatkan detail data pada citra yang digunakan. Lalu yang kedua adalah *library* CUDA, sudah tentu *library* ini disertakan untuk memakai fungsi-fungsi CUDA dalam memanfaatkan GPU untuk mempercepat proses komputasi. Ketiga yang sudah umum adalah *library* bahasa C, yang digunakan untuk mendukung segala sesuatu dalam membuat program ini. Lalu terakhir adalah sebuah baris code dengan definisi I2D yang berfungsi menentukan indeks untuk mengkonversi data citra yang berbentuk matriks 2D ke dalam bentuk matriks 1D.

Setelah *library* dideklarasikan, barulah dilanjutkan dengan mendeklarasikan variable-variable yang akan digunakan. Terdapat 2 jenis variable, yaitu yang bersifat global dan lokal.

```
//variable global
int MAXiter = 0;
int proses  = 0;
```

```
int metode  = 0;
float total_waktu = 0;

//variable lokal
int menu = 0;
char file_src[100];
char file_brk[100];
char file_msk[100];

Mat asli;
Mat image;
Mat mask;

Mat Simpan;
Mat RESULT;

int CHNL;
int baris = 0;
int kolom = 0;
int count = 0;
```

Kode 3.2. Deklarasi Variable

Variable global yang digunakan ada 4 buah, yaitu MAXiter untuk menyimpan jumlah iterasi, proses untuk menyimpan pilihan prosesor, metode untuk menyimpan pilihan metode yang digunakan, dan terakhir total_waktu untuk menyimpan total waktu hasil proses komputasi dalam *millisecond*. Lalu ada variable local yang berfungsi untuk menyimpan data citra, mulai dari inputan nama citra dan data detail citra dalam tipe data Mat, beserta beberapa data integer untuk menyimpan jumlah baris, kolom, dan *channel* pada citra.

Selain variable, terdapat beberapa prosedur dan fungsi yang dibuat untuk membuat program menjadi lebih rapi dan mudah dipahami.

```
//--Prosesdur Pemroses PERONA-MALIK
void PeronaMalikGPU(double *h_in, double *h_out, int elems, int rows, int cols,
double *lambda, double *G, int metode);
```

```
void PeronaMalikCPU(double *h_in, double *h_out, int elems, int rows, int cols,
double *lambda, double *G, int metode);

__global__ void device_COPY_DATA(double *d_in, double *d_out);
__global__ void device_PERONA_MALIK(double *d_in, double *d_out, int M, int N,
double *lambda, double *G, int metode);
          void host_PERONA_MALIK(double *d_in, double *d_out, int M, int N,
double *lambda, double *G, int metode);

//--Inisialisasi Variable
void inisialisasi(Mat &image, Mat &mask, Mat &lambda, Mat &g, Mat &u);

//--Image Validation - cek kualitas image
void getMSE_PSNR(Mat I1, Mat I2); // I1-source , I2-result
Scalar getSSIM(Mat I1, Mat I2);  // I1-source , I2-result
```

Kode 3.3. Prosedur dan Fungsi

Pada baris kode diatas, dipaparkan beberapa prosedur, mulai dari prosedur PeronaMalik yang diproses dalam CPU dan GPU, prosedur inisialisasi untuk menormaliasasi citra agar dapat diolah lebih lanjut, dan prosedur untuk memvalidasi citra. Disetiap prosedur terdapat parameter-parameter yang berfungsi meminta nilai dari program sesuai dengan nama dari parameter tersebut. Isi dari prosedur dan fungsi akan dijelaskan lebih lanjut pada bagian berikutnya.

Jika semua variable dan prosedur telah dideklarasikan maka selanjutnya dilakukan proses load citra, proses ini menggunakan fungsi **imread** pada openCV. Fungsi **imread** memiliki beberapa parameter, namun secara umum untuk load citra hanya digunakan satu parameter, yaitu nama dari citra yang hendak diload beserta format citra tersebut.

```
//--for graysclae
Mat temp_asli  = imread(file_src, CV_LOAD_IMAGE_GRAYSCALE);
Mat temp_image = imread(file_brk, CV_LOAD_IMAGE_GRAYSCALE);
```

```
Mat temp_mask  = imread(file_msk, CV_LOAD_IMAGE_GRAYSCALE);

//--for color
Mat temp_asli  = imread(file_src);
Mat temp_image = imread(file_brk);
Mat temp_mask  = imread(file_msk, CV_LOAD_IMAGE_GRAYSCALE);


temp_asli.copyTo(asli);
temp_image.copyTo(image);
temp_mask.copyTo(mask);


//membuat sebuah window dg nama My Image Scratch
cv::namedWindow("My Image Scratch");
//menampilkan gambar yang tersimpan di variable ke window
cv::imshow("My Image Scratch",image);
cvMoveWindow("My Image Scratch", 0, 0);
cv::waitKey(2);

//membuat sebuah window dg nama My Image Mask
cv::namedWindow("My Image Mask");
//menampilkan gambar yang tersimpan di variable ke window
cv::imshow("My Image Mask",mask);
cvMoveWindow("My Image Mask", 0, 150);
cv::waitKey(2);
```

Kode 3.4. load and display image

Terdapat tiga buah variable temp untuk masing-masing citra yang akan di load. Sebelum citra dimasukkan pada variable yang sebenarnya(asli, image, mask), pertama kali load digunakan variable temp, tujuannya agar ada *backup* citra saat terjadi masalah pada citra yang sebenarnya. Pada parameter **imread** terdapat nama variable file_src, file_brk, dan file_msk, itu adalah variable bertipe string yang berisi nilai nama citra yang didapat dari inputan pengguna. Khusus untuk citra *mask* dan citra *grayscale*, menggunakan dua buah parameter, yang pertama nama citra dan yang kedua fungsi konversi citra secara langsung kedalam bentuk *grayscale* dengan satu *channel*. Setelah

berhasil load image, citra tersebut lalu ditampilkan pada sebuah window. Untuk menampilkan citra digunakan fungsi **imshow** yang dimiliki oleh OpenCV. Fungsi **imshow** memiliki 2 parameter secara umum, yaitu nama window yang akan dibuat, dan variable yang menyimpan data citra yang sudah diload.

III.3.3.2. Implementasi Algoritma Perona-Malik

Sebelumnya telah sedikit dijelaskan implementasi algoritma secara umum, mulai dari proses deklarasi variable hingga membaca citra dan menampilkannya. Sekarang akan dilanjutkan dengan implementasi algoritma pada proses utama, yaitu *inpainting*. Pertama-tama sebelum masuk ke proses utama, dideklaraiskan juga beberapa variable tambahan yang dibutuhkan, seperti yang terlihat pada baris code dibawah.

```
// proses untuk konversi gambar dari tipe uchar 8 bit dan
// 1 channel ke tipe float 64 bit 1 channel
image.convertTo(image, CV_64FC1);
mask.convertTo(mask, CV_64FC1);

// mendeklarasikan variable untuk pendukung proses inpainting
double *temp_img    = new double[baris*kolom];
double *temp_lambda = new double[baris*kolom];
double *temp_G      = new double[baris*kolom];
cv::Mat lambda, g, u;

// inisialisasi setiap variable yang digunakan
inisialisasi(image, mask, lambda, g, u);
```

Kode 3.5. inisialisasi

Variable image dan mask yang menyimpan data citra yang awalnya bertipe data uchar(8U) dikonversi ke tipe

data double(64F) untuk proses normalisasi pada bagian selanjutnya. Lalu dibuatkan tiga buah variable dengan pointer bertipe data double dengan panjang indeks array sesuai dengan jumlah piksel citra. Variable-variable tersebut adalah temp_img, temp_lambda, dan temp_G, ketiga variable tersebut akan digunakan untuk menyimpan data citra kedalam data array 1 dimensi. Lalu prosedur inisialisasi dipanggil yang berisi parameter image, mask, lambda, g, dan u. Prosedur ini digunakan untuk proses normaliasi citra.

Proses normalisasi terdapat didalam prosedur inisialisasi seperti ditunjukan pada kode diabawah. Prosedur memiliki beberapa parameter *input output* seperti yang telah disebutkan diatas. Parameter inilah yang nilainya akan digunakan untuk proses *inpainting*.

```
void inisialisasi(Mat &image, Mat &mask, Mat &lambda, Mat &g, Mat &u)
{
        double maxS=0, minS=0, maxM=0, minM=0, maxG=0, minG=0;

        minMaxLoc(image,&minS,&maxS);
        minMaxLoc(mask,&minM,&maxM);
        image = image/maxS; //normalisasi
        mask = 1-mask/maxM; //membuat gambar negatif

        lambda = mask.clone();
        g = image.clone();

        minMaxLoc(g,&minG,&maxG);
        divide(g,maxG,g);

        double lambda0=10;
        lambda = lambda.mul(lambda0);

        g = image.clone();
        u = image.clone();
        u.convertTo(u, CV_64FC1);
}
```

Kode 3.6. Normalisasi

Didalam prosedur terdapat beberapa variable lagi yang dideklarasikan, yaitu maxS, minS, dan lainnya. Variable tersebut digunakan untuk menyimpan nilai tertinggi dan nilai terendah dari citra yang dideteksi oleh fungsi **minMaxLoc**. Setelah mendapatkan nilai tertinggi dari image dan mask, makan nilai yang ada disetiap piksel pada variable image dan mask dibagi dengan nilai tertinggi dimasing-masing variable. Tujuannya adalah untuk menormalisasi, yaitu membentuk nilai yang seimbang dari range 0 hingga 1 dalam tipe data double pada citra yang dioperasikan. Jika variable image dan mask sudah dinormalisasi, lalu buat *cloning*nya ke variable g dan u, setelah itu setiap variable siap digunakan pada program untuk proses *inpainting*.

Ketika semua variable yang hendak digunakan sudah siap untuk diproses, maka dapat dilanjutkan ke proses *inpainting*. Sebelum itu, semua data citra yang hendak digunakan dikonversi terlebih dulu kedalam variable array 1 dimensi bertipe double.

```
// proses pemindahan nilai dari variable inisialisasi ke variable temp
for(int y=0; y<kolom; y++)
{
         for(int x=0; x<baris; x++)
         {
                  int i1;
                  i1 = y+x*kolom;
                  temp_img[i1]    = u.at<double>(x,y);
                  temp_lambda[i1] = lambda.at<double>(x,y);
                  temp_G[i1]      = g.at<double>(x,y);
         }
}

double *h_in  = temp_img;
double *h_out = temp_img;
```

Kode 3.7. Konversi image ke array 1D

Tujuan dari konversi citra ke array 1 dimensi adalah untuk membuat pengaksesan data per pikselnya menjadi lebih mudah dan lebih cepat. Untuk posisi data pada array, tiap indeksnya diposisikan sama sesuai dengan posisi data saat masih menjadi 2 dimensi, maka digunakan rumus `i1 = y+x*kolom;` untuk membentuk indeks tersebut. Persiapan terakhir adalah membuat dua buah variable yang bernama h_in dan h_out. Variable h_in di*assign* nilai dari dari variable temp_img, yang nantinya digunakan sebagai data *input* untuk proses *inpainting*. Begitu juga dengan h_out, tetapi fungsi h_out adalah untuk menyimpan hasil akhir proses *inpainting*. Setelah semua proses diatas, barulah dapat dilanjutkan ke proses *inpainting*.

Untuk proses *inpainting* akan digunakan dua buah prosedur yang telah sempat disebutkan diatas, yaitu prosedur pemroses pada CPU dan pada GPU.

```
// Prosedur PM CPU
PeronaMalikCPU(h_in, h_out, baris, kolom, temp_lambda, temp_G, metode);

// Prosedur PM GPU
PeronaMalikGPU(h_in, h_out, baris, kolom, temp_lambda, temp_G, metode);
```

Kode 3.8. Prosedur inpainting

III.3.3.2.1. Implementasi Algoritma Perona-Malik CPU

Algoritma Perona-Malik pada CPU menggunakan dua buah prosedur yang berbeda, prosedur pertama digunakan

untuk melakukan iterasi beserta menghitung waktu komputasi, dan prosedur kedua untuk memproses citra.

```
void PeronaMalikCPU(double *h_in, double *h_out, int rows, int cols, double
*lambda, double *G, int metode)
{

          //timer
          clock_t start, stop, selisih;
          float elapsedTime;

          start = clock();

          for(int zz=0; zz<MAXiter; zz++)
          {
          host_PERONA_MALIK(h_in, h_out, rows, cols, lambda, G, metode);
          h_in = h_out;
          }

          stop = clock();

          selisih = stop - start;
          elapsedTime = selisih / (float)CLOCKS_PER_SEC;

          printf( "\nTime to generate CPU :  %3.3f ms", elapsedTime*1000 );
          total_waktu = total_waktu + (elapsedTime*1000);

}
```

Kode 3.9. Prosedur PeronaMalikCPU

Pada kode 3.9, kode tersebut adalah prosedur yang pertama, prosedur berisi tiga buah variable bertipe *clock* yang digunakan untuk mencatat waktu mulai, selesai dan selisih waktu komputasi. Selanjutnya terdapat sebuah perulangan yang berulang hingga maksimun iterasi sesuai dengan inputan pengguna. Dalam perulangan dipanggil prosedur yang kedua, yaitu prosedur yang memproses citra, proses yang dilakukan adalah proses *inpainting*. Semua parameter pada prosedur

pertama dimasukkan sebagai parameter di prosedur yang kedua, dimana parameter tersebut diantaranya adalah h_in yang berperan sebagai inputan citra, h_out yang berperan sebagai hasil *inpainting*, rows yang berperan sebagai jumlah baris, cols yang berperan sebagai jumlah kolom, lambda yang berperan sebagai *masking*, G yang berperan sebagai citra statis, dan metode yang berperan sebagai pilihan metode yang digunakan. Setelah iterasi pertama selesai dilakukan, maka untuk melakukan iterasi selanjutnya, nilai pada h_in haruslah di*update* dengan nilai hasil *inpainting* yang ada pada h_out. Dengan begitu hasil *inpainting* akan terlihat pada setiap iterasinya. Lama waktu komputasi dihitung berdasarkan selisih antara waktu selesai iterasi dengan waktu mulai iterasi. Setelah waktu komputasi dicatat, maka akan disimpan pada variable total_waktu, dan nantinya akan ditampilkan pada program.

Masuk pada prosedur yang kedua, yaitu prosedur untuk memproses citra.

```
void host_PERONA_MALIK(double *d_in, double *d_out, int M, int N, double
*lambda, double *G, int metode)
{

          double deltaN;
          double deltaS;
          double deltaW;
          double deltaE;

          double cN, cS, cW, cE;
          double dt=0.1,K=10;

          int index = 0;
          int NN,SS,EE,WW;
          int left,right,top,bottom;

          for(int y=0; y<N; y++)
```

```
        {
                for(int x=0; x<M; x++)
                {

                        NN=y-1;
                        SS=y+1;
                        EE=x-1;
                        WW=x+1;

                        if(y==0) NN=2;              left   = I2D(N,x,NN);
                        if(y==N-1) SS=N-2;          right  = I2D(N,x,SS);
                        if(x==0) EE=2;              top    = I2D(N,EE,y);
                        if(x==M-1) WW=M-2;          bottom = I2D(N,WW,y);

                        deltaN = d_in[left]-d_in[index];
                        deltaS = d_in[right]-d_in[index];
                        deltaE = d_in[top]-d_in[index];
                        deltaW = d_in[bottom]-d_in[index];

                        if(metode == 3)
                        {
                                cN = 1.0;
                                cS = 1.0;
                                cE = 1.0;
                                cW = 1.0;
                        }
                        else if(metode == 1)
                        {

                                cN = exp(-(pow((deltaN/K),2)));
                                cS = exp(-(pow((deltaS/K),2)));
                                cE = exp(-(pow((deltaE/K),2)));
                                cW = exp(-(pow((deltaW/K),2)));
                        }
                        else if(metode == 2)
                        {

                                cN = 1.0/(1.0+pow((deltaN/K),2));
                                cS = 1.0/(1.0+pow((deltaS/K),2));
                                cE = 1.0/(1.0+pow((deltaE/K),2));
                                cW = 1.0/(1.0+pow((deltaW/K),2));
                        }

                double dummy = lambda[index]*(G[index] - d_in[index]);

                d_out[index] = d_in[index] + dt * (cN*deltaN + cS*deltaS +
                 cW*deltaW + cE*deltaE + dummy);

                }
        }
```

```
}
```

Kode 3.10. Prosedur Proses PeronaMalikCPU

Pada prosedur dideklarasikan variable delta dengan fungsi untuk menyimpan nilai selisih antara indeks yang dioperasikan dengan indeks sekitarnya. Lalu ada variable c, dt, dan K untuk mendukung rumus utama inpainting. Ada empat buah variable left, right, top, dan bottom untuk menyimpan nilai indeks array sebelum nilai piksel pada indeks tersebut diakses oleh variable lain. Setelah itu dibuat *nested loop* yaitu perulangan dalam perulangan yang isinya adalah rumus untuk memproses citra *inpainting*. Perulangan digunakan karena pada CPU proses yang berjalan adalah secara serial. Jadi proses dilaksanakan satu per satu baris code dari atas ke bawah.

Pertama ditentukan nilai indeks disekitar piksel yang dioperasikan dengan memanfaatkan variable perulangan, dimana untuk mengakses indeks disebelah kanan dan kiri piksel digunakan x±1, dan untuk atas dan bawah piksel digunakan y±1. Setelah itu dilakukan pengecekan terhadap nilai x dan y, apakah nilainya ada yang kurang dari indeks array terendah atau lebih dari indeks array tertinggi. Hal tersebut dilakukan untuk mencegah kesalahan pengaksesan indeks pada array. Kemudian untuk mendapatkan posisi piksel yang sebenarnya pada array digunakan fungsi I2D dengan parameter kordinat x y yang sudah ditentukan sebelumnya. Jika indeks-indeks yang diperlukan sudah ada, maka dapat dicari selisih antara piksel yang

dioperasikan dengan piksel disekitarnya dan kemudian disimpan pada variable delta.

Selanjutnya masuk pada metode, hasil perhitungan metode akan disimpan pada variable c. Metode yang disediakan ada tiga jenis, yaitu metode linier(heat), metode Perona-Malik 1, dan Perona-Malik 2. Untuk metode linier(heat) setiap nilai c diset 1.0, dan untuk metode Perona-Malik masing-masing metode memiliki rumus yang berbeda dan tentunya nilai yang berbeda pula. Rumus Perona-Malik didapatkan berdasarkan hasil persamaan difusi Perona-Malik atas nilai $c_x = c_x$ = fungsi $I(x,y)$, lalu ditemukanlah hasil diskretisasi menjadi Perona-Malik 1 dan Perona-Malik 2. Untuk Perona-Malik 1 persamaannya adalah $c_x = \exp(-(pow((delta_x/K),2)))$, setiap nilai delta dibagi dengan K, dan setelah itu nilai hasil bagi dikuadratkan agar hasilnya menjadi positif, kemudian dilakukan proses exponensial. Begitu juga dengan Perona-Malik 2, persamaannya $c_x = 1.0/(1.0+pow((delta_x/K),2))$, hanya saja hasil dari pengkuadratannya ditambah dengan nilai 1.0 dan kemudian dipangkatkan -1.0.

Semua variable telah siap, mulai dari citra yang akan diproses, *masking*, delta, c, dan lainnya maka citra siap di*inpaint*. Sebelum itu, tentukan area yang akan di*inpaint* dengan mengalikan *masking* dan citra terkini(citra rusak) lalu disimpan pada variable dummy. Kemudian *update* nilai d_out disetiap pikselnya dengan rumus d_in ditambah dt lalu dikali dengan hasil deteksi tepi (c * delta) yang telah ditambahkan dengan dummy,

maka proses *inpainting* akan berjalan piksel demi piksel disetiap perulanagn dan disetiap iterasinya.

III.3.3.2.2. Implementasi Algoritma Perona-Malik GPU

Sama seperti algoritma Perona-Malik pada CPU, Algoritma Perona-Malik pada GPU tidak banyak berbeda, pada GPU memanfaatkan tiga buah prosedur, prosedur pertama digunakan untuk melakukan alokasi memori pada GPU dan iterasi beserta menghitung waktu komputasi, prosedur kedua untuk memproses citra, dan prosedur ketiga untuk transfer data antar memori pada GPU.

```
void PeronaMalikGPU(double *h_in, double *h_out, int rows, int cols, double
*lambda, double *G, int metode)
{
         double *d_in;
         double *d_out;
         double *d_lambda;
         double *d_G;

         dim3    blok(16,16);
         dim3    grid(cols/16,rows/16);

         cudaEvent_t     start, stop;
         float           elapsedTime;

        cudaEventCreate( &start  );
        cudaEventCreate( &stop  );

         cudaSetDevice(0);

         cudaMalloc((void**) &d_in, rows*cols*sizeof(double));
         cudaMalloc((void**) &d_out, rows*cols*sizeof(double));
         cudaMalloc((void**) &d_lambda, rows*cols*sizeof(double));
         cudaMalloc((void**) &d_G, rows*cols*sizeof(double));

         cudaMemcpy(d_in, h_in, rows*cols*sizeof(double),
cudaMemcpyHostToDevice);
         cudaMemcpy(d_out, h_out, rows*cols*sizeof(double),
cudaMemcpyHostToDevice);
         cudaMemcpy(d_lambda, lambda, rows*cols*sizeof(double),
cudaMemcpyHostToDevice);
         cudaMemcpy(d_G, G, rows*cols*sizeof(double), cudaMemcpyHostToDevice);

        cudaEventRecord( start, 0 );
```

```
        for(int zz=0; zz<MAXiter; zz++)
        {
                 device_PERONA_MALIK<<<grid,blok>>>(d_in, d_out, rows, cols,
                  d_lambda, d_G, metode);
                 device_COPY_DATA<<<grid,blok>>>(d_in, d_out);
        }

        cudaMemcpy(h_out, d_out, elems*sizeof(double),
        cudaMemcpyDeviceToHost);
        cudaDeviceSynchronize();

        cudaEventRecord( stop, 0 );
        cudaEventSynchronize( stop );

        cudaEventElapsedTime( &elapsedTime, start, stop );

        printf( "\nTime to generate GPU :  %3.3f ms", elapsedTime );
        total_waktu = total_waktu + (elapsedTime);

        cudaEventDestroy( start );
        cudaEventDestroy( stop );

        cudaFree(d_in);
        cudaFree(d_out);
        cudaFree(d_lambda);
        cudaFree(d_G);
}
```

Kode 3.11. Prosedur PeronaMalikGPU

Pada kode 3.11, kode tersebut adalah prosedur yang pertama, pada awal prosedur dideklarasikan empat buah variable dengan pointer yang akan digunakan untuk menampung nilai data citra pada parameter sebelum nanti ditransfer ke memori pada GPU. Variable tersebut adalah d_in, d_out, d_lambda, dan d_G. Kemudian tentukan jenis dan jumlah *thread* maupun *block* yang hendak digunakan pada GPU. *Thread* disini adalah tempat data akan diproses untuk setiap satuan pikselnya, lalu *block* adalah kumpulan dari beberapa *thread*, dan *grid* adalah kumpulan dari beberapa *block* baik yang disusun secara 1 dimensi atau lebih. Pada program ini untuk setiap *block* akan disediakan 16x16 *thread*, dan untuk setiap *grid*

akan disediakan jumlah kolom citra dibagi 16 x jumlah baris citra dibagi 16. Hal tersebut dilakukan agar memori yang digunakan untuk memproses data bekerja secara optimal, jadi tidak ada memori yang *idle*. Untuk menghitung waktu komputasi dideklarasikan dua buah variable bertipe **cudaEvent_t** yang fungsinya sama dengan *clock* pada CPU.

cudaMalloc adalah sebuah fungsi yang digunakan untuk mengalokasikan memori pada GPU(device), ada 4 buah variable dengan pointer yang akan dialokasikan, yaitu empat variable yang dideklarisakn paling atas pada kode 3.11. Jumlah memori yang dialokasikan adalah sebanyak jumlah kolom x jumlah baris citra dalam ukuran *byte*. Setelah pengalokasian memori, lalu dilanjutkan dengan pemindahan data citra yang awalnya berada pada CPU ke memori yang sudah dialokasikan di GPU menggunakan fungsi **cudaMemcpy**. Fungsi **cudaMemcpy** memiliki empat buah atribut yaitu pertama variable pointer tujuan transfer, kedua variable asal transfer, ketiga banyak memori yang akan digunakan, dan yang keempat adalah metode transfer, apakah akan dilakukan transfer dari CPU(host) ke GPU(device) atau sebaliknya.

Selanjutnya terdapat sebuah perulangan yang berulang hingga maksimun iterasi sesuai dengan inputan pengguna. Dalam perulangan dipanggil prosedur yang kedua dan ketiga, yaitu prosedur yang memproses citra dan prosedur transfer data GPU , proses yang dilakukan adalah proses *inpainting*. Semua variable yang telah dialokasikan pada memori GPU dimasukkan sebagai

parameter di prosedur yang kedua, dimana parameter tersebut daintaranya adalah d_in yang berperan sebagai inputan citra, d_out yang berperan sebagai hasil *inpianting*, rows yang berperan sebagai jumlah baris, cols yang berperan sebagai jumlah kolom, d_lambda yang berperan sebagai masking, d_G yang berperan sebagai citra statis, dan metode yang berperan sebagai pilihan metode yang digunakan. Setelah iterasi pertama selesai dilakukan, maka untuk melakukan iterasi selanjutnya, nilai pada d_in haruslah di*update* dengan nilai hasil *inpainting* yang ada pada d_out. Dengan begitu hasil *inpainting* akan terlihat pada setiap iterasinya, maka dari itu digunakanlah prosedur yang ketiga yaitu prosedur transfer data GPU.

Apabila iterasi sudah selesai dilakukan, data hasil proses *inpainting* dikirimkan lagi dari GPU ke CPU menggunakan **cudaMemcpy** dengan metode yang sebaliknya yaitu dari device ke host. Terakhir bersihkan memori pada GPU yang telah dialokasi sebelumnya menggunakan fungsi **cudaFree** dan disertakan parameter berupa variable yang dipakai saat proses *inpainting* pada GPU. Lama waktu komputasi dihitung berdasarkan selisih antara waktu selesai iterasi dengan waktu mulai iterasi. Setelah waktu komputasi dicatat, maka akan disimpan pada variable total_waktu, dan nantinya akan ditampilkan pada program.

Masuk pada prosedur yang kedua, yaitu prosedur untuk memproses citra.

```
__global__ void device_PERONA_MALIK(double *d_in, double *d_out, int M, int N,
double *lambda, double *G, int metode)
{

          double deltaN;
          double deltaS;
          double deltaW;
          double deltaE;

          double cN, cS, cW, cE;
          double dt = 0.1;
          int K = 10;

    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;
    int index = x + y * blockDim.x * gridDim.x;

    int left = index - 1;
    int right = index + 1;
    if (x < 2)   left++;
    if (x == M-1) right--;

    int top = index - N;
    int bottom = index + N;
    if (y < 2)   top += N;
    if (y == N-1) bottom -= N;


                  deltaN = d_in[left]-d_in[index];
                  deltaS = d_in[right]-d_in[index];
                  deltaE = d_in[top]-d_in[index];
                  deltaW = d_in[bottom]-d_in[index];

          if(metode == 3)
          {
                  cN = 1.0;
                  cS = 1.0;
                  cE = 1.0;
                  cW = 1.0;
          }
          else if(metode == 1)
          {

                  cN = exp(-(pow((deltaN/K),2)));
                  cS = exp(-(pow((deltaS/K),2)));
                  cE = exp(-(pow((deltaE/K),2)));
                  cW = exp(-(pow((deltaW/K),2)));
          }
          else if(metode == 2)
          {
```

```
                    cN = 1.0/(1.0+pow((deltaN/K),2));
                    cS = 1.0/(1.0+pow((deltaS/K),2));
                    cE = 1.0/(1.0+pow((deltaE/K),2));
                    cW = 1.0/(1.0+pow((deltaW/K),2));
          }

          double dummy = lambda[index]*(G[index] - d_in[index]);

          d_out[index] = d_in[index] + dt * (cN*deltaN + cS*deltaS + cW*deltaW +
cE*deltaE + dummy);

}
```

Kode 3.12. Prosedur Proses PeronaMalikGPU

Prosedur proses PeronaMalikGPU ini hampir sama seperti prosedur proses PeronaMalikCPU, yang membedakan adalah pada CPU menggunakan *nested loop* untuk memproses tapi pada GPU menggunakan *thread* dan dieksekusi secara paralel. Pada prosedur dideklarasikan variable delta dengan fungsi untuk menyimpan nilai selisih antara indeks yang dioperasikan dengan indeks sekitarnya. Lalu ada variable c, dt, dan K untuk mendukung rumus utama inpainting. Ada empat buah variable left, right, top, dan bottom untuk menyimpan nilai indeks array sebelum nilai pada indeks tersebut diakses oleh variable lain. Setelah itu dideklarasikan variable x dan y untuk menentukan indeks pengakses data berdasarkan *thread, block*, dan *grid*. Untuk menentukan indeks yang akan dioperasikan digunakan rumus x + y * blockDim.x * gridDim.x ,dimana x dan y adalah penentu indeks disetiap piksel, blockDim.x adalah panjang dimensi block pada kordinat x, dan gridDim.x adalah panjang dimensi grid pada kordinat x. Setiap indeks yang ditentukan sekaligus juga akan langsung dioperasikan, hal tersebut karena komputasi pada GPU berjalan secara

paralel. Jadi semua data piksel dieksekusi secara bersamaan, dan oleh karena itu proses paralel ini tidak membutuhkan perulangan.

Pertama ditentukan nilai indeks disekitar piksel yang dioperasikan dengan memanfaatkan variable index, dimana untuk mengakses indeks disebelah kanan dan kiri piksel digunakan index±1, dan untuk atas dan bawah piksel digunakan index±N (N adalah panjang kolom). Setelah itu dilakukan pengecekan terhadap nilai x dan y, apakah nilainya ada yang kurang dari indeks array terendah atau lebih dari indeks array tertinggi. Hal tersebut dilakukan untuk mencegah kesalahan pengaksesan indeks pada array. Jika indeks-indeks yang diperlukan sudah ada, maka dapat dicari selisih antara piksel yang dioperasikan dengan piksel disekitarnya dan kemudian disimpan pada variable delta.

Selanjutnya masuk pada metode, hasil perhitungan metode akan disimpan pada variable c. Metode yang disediakan ada tiga jenis, yaitu metode linier(heat), metode Perona-Malik 1, dan Perona-Malik 2. Untuk metode linier(heat) setiap nilai c diset 1.0, dan untuk metode Perona-Malik masing-masing metode memiliki rumus yang berbeda dan tentunya nilai yang berbeda pula. Rumus Perona-Malik didapatkan berdasarkan hasil persamaan difusi Perona-Malik atas nilai c_x = c_x = fungsi $I(x,y)$, lalu ditemukanlah hasil diskretisasi menjadi Perona-Malik 1 dan Perona-Malik 2. Untuk Perona-Malik 1 persamaannya adalah $c_x = \exp(-(\text{pow}((\text{delta}_x/K),2)))$, setiap nilai delta dibagi dengan K, dan stelah itu nilai hasil bagi

dikuadratkan agar hasilnya menjadi positif, kemudian dilakukan proses exponensial. Begitu juga dengan Perona-Malik 2, persamaannya $c_x = 1.0/(1.0+pow((delta_x/K),2))$, hanya saja hasil dari pengkuadratannya ditambah dengan nilai 1.0 dan kemudian dipangkatkan -1.0.

Semua variable telah siap, mulai dari citra yang akan diproses, *masking*, delta, c, dan lainnya maka citra siap di*inpaint*. Sebelum itu, tentukan area yang akan di*inpaint* dengan mengalikan *masking* dan citra terkini(citra rusak) lalu disimpan pada variable dummy. Kemudian *update* nilai d_out disetiap pikselnya dengan rumus d_in ditambah dt lalu dikali dengan hasil deteksi tepi (c * delta) yang telah ditambahkan dengan dummy, maka proses inpainting akan berjalan secara bersamaan disetiap piksel dan disetiap iterasinya.

III.3.3.3. Implementasi Algoritma Validasi Image

Untuk melakukan validasi citra dibutuhkan 2 buah citra, yaitu citra awal sebelum diproses dan citra akhir yang sudah diproses. Citra awal sudah tersedia yaitu citra yang tersimpan pada variable asli dengan tipe uchar 8 bit. Citra akhir juga sudah tersedia dari hasil proses inpainting, tetapi tipenya adalah array double 64 bit, berbeda dengan citra awal, oleh karena itu citra akhir ini harus dikonversi dahulu dan disimpan pada variable lain untuk membuatnya menjadi tipe uchar 8 bit. Untuk konversi tersebut, kodenya seperti kode yang ada dibawah.

```
// membentuk kembali gambar dari hasil proses inpainting
Simpan.convertTo(Simpan, CV_8UC1);

for(int y=0; y<kolom; y++)
{
         for(int x=0; x<baris; x++)
         {
                  int i1;
                  i1 = y+x*kolom;

                  Simpan.at<uchar>(x,y) = (int)(h_out[i1]*255);
         }

}
```

Kode 3.13. Konversi Array ke Image

Pertama konversi variable Simpan kedalam tipe uchar 8 bit, dimana variable inilah yang akan menyimpan data gambar hasil konversi. Kemudian dibuatkan *nested loop* berdasarkan lebar dan panjang citra. Lalu didalam perulangan dideklarasikan variable dengan berisi nilai `y+x*kolom`; , nilai tersebut adalah untuk menentukan indeks data secara 2 dimensi dari array 1 dimensi. Hal tersebut dilakukan karena data yang akan dikonversi berasal dari array 1 dimensi bertipe double, dan akan dimasukan ke citra 2 dimensi bertipe uchar. Setelah itu untuk memasukan nilainya digunakan kode seperti ini `Simpan.at<uchar>(x,y) = (int)(h_out[i1]*255);` , data pada array 1 dimensi dikalikan dengan nilai 255, agar setiap nilai yang dimasukkan ke variable Simpan berada pada range 0 hingga 255. Sehingga untuk validasi nanti sudah tersedia dua buah citra dengan tipe yang sama.

Proses validasi akan dilakukan dengan tiga cara validasi, pertama validasi dengan *Mean Squared Error* (MSE), kedua dengan *Peak Signal-to-Noise Ratio* (PSNR),

dan yang ketiga *Structural SIMilarity* (SSIM) (Varnan et al., 2011).

III.3.3.3.1. Implementasi Algoritma Validasi MSE

Validasi citra dengan MSE dilakukan dengan cara mencari rata-rata perbedaan diantara dua buah citra, yaitu citra awal sebelum *inpainting* dan citra akhir setelah *inpainting*. Algoritma yang digunakan adalah sebagai berikut.

```
void getMSE(Mat I1, Mat I2)
{
        I1.convertTo(I1, CV_64F);
        I2.convertTo(I2, CV_64F);

    Mat s1;
    absdiff(I1, I2, s1);       // |I1 - I2|
    s1.convertTo(s1, CV_64F);  // cannot make a square on 8 bits
    s1 = s1.mul(s1);           // |I1 - I2|^2

    Scalar s = sum(s1);        // sum elements per channel

    double sse = s.val[0] + s.val[1] + s.val[2]; // sum channels

        if( sse <= 1e-10) // for small values return zero
        {
                printf("MSE   : %.2f \n",0);
        }
        else
        {
                //MSE
                double mse  = sse / (double)(I1.channels() * I1.total());
                printf("MSE   : %.2f \n",mse);
        }
}
```

Kode 3.14. Validasi MSE

Pada algoritma atau prosedur diatas, kedua citra baik citra awal dan akhir, dikonversi ke tipe double

64 bit. Lalu dilakukan pengurangan antara citra akhir dengan citra awal untuk setiap pikselnya, dan hasil selisihnya diabsolutkan agar hasilnya menjadi positif, hal tersebut dapat dilakukan langsung dengan fungsi **absdiff**. **Absdiff** memiliki 3 buah parameter, dua parameter pertama adalah citra yang dioperasikan, dan parameter ketiga adalah untuk menyimpan hasil operasi. Kemudian hasil selisih tadi dikonversi lagi ke tipe double 64 bit, hal tersebut dilakukan karena tipe uchar 8 bit tidak dapat dikuadratkan pada langkah selanjutnya. Barulah hasil selisih dikuadratkan atau dikalikan dengan dirinya sendiri. Setelah itu jumlahkan semua data piksel dengan fungsi **sum**, dan hasilnya disimpan pada variable s bertipe Scalar. Jika semua elemen piksel sudah dijumlahkan, maka hasil penjumlahan tersebut dibagi dengan jumlah pikselnya(lebar x panjang citra), maka didapatkanlah rata-rata perbedaan pada citra tersebut. Jadi semakin kecil *error* yang dihasilkan, maka semakin bagus juga kualitas citra setelah diproses.

III.3.3.3.2. Implementasi Algoritma Validasi PSNR

Validasi citra dengan PSNR dilakukan dengan perhitungan yang sama seperti Validasi MSE, tetapi ditambah sedikit perhitungan lagi. Perhitungan yang dimaksud adalah seperti rumus dibawah.

```
        //PSNR
        double psnr = 10.0 * log10((255 * 255) / mse);

        printf("PSNR  : %.2f \n",psnr);
```

Kode 3.15. Validasi PSNR

Validasi PSNR adalah mencari rasio diantara sinyal pada citra awal dengan sinyal pada citra akhir yang telah diproses. Untuk mendapatkan rasio, 10.0 dikalikan dengan log10 dengan parameter nilai 255*255 dibagi dengan nilai MSE yang sudah dihitung sebelumnya. Semakin besar rasio yang didapatkan, maka semakin bagus juga kulaitas citranya. Maksudnya adalah kualitas citra yang sudah diproses semakin mendekati kualitas citra awal yaitu citra asli yang belum diproses.

III.3.3.3.3. Implementasi Algoritma Validasi SSIM

Validasi citra dengan SSIM berbasis *single scale image*, algoritma SSIM mengadaptasi sistem visual manusia untuk memproses informasi struktural dari citra. Algoritma SSIM mengukur perubahan informasi citra, baik itu berupa kontras dan struktur dari citra yang dibandingkan (Rouse & Hemami, 2008). Hasil validasi berupa prosentase, Jadi semakin hasil validasi mendekati 100%, maka semakin bagus kualitas citra yang dihasilkan dari proses *inpainting*. Dilihat dari hasil pengukuran kualitas citra, validasi SSIM lebih baik dari MSE dan PSNR.

```
Scalar getSSIM( Mat I1,  Mat I2)
{
```

```
const double C1 = 6.5025, C2 = 58.5225;
/***************************** INITS *********************************/
int d     = CV_64F;

//Mat I1, I2;
I1.convertTo(I1, d);           // cannot calculate on one byte large values
I2.convertTo(I2, d);

Mat I2_2   = I2.mul(I2);        // I2^2
Mat I1_2   = I1.mul(I1);        // I1^2
Mat I1_I2  = I1.mul(I2);        // I1 * I2

/*****************************PRELIMINARY COMPUTING ******************************/

Mat mu1, mu2;   //
GaussianBlur(I1, mu1, Size(11, 11), 1.5);
GaussianBlur(I2, mu2, Size(11, 11), 1.5);

Mat mu1_2   =   mu1.mul(mu1);
Mat mu2_2   =   mu2.mul(mu2);
Mat mu1_mu2 =   mu1.mul(mu2);

Mat sigma1_2, sigma2_2, sigma12;

GaussianBlur(I1_2, sigma1_2, Size(11, 11), 1.5);
sigma1_2 -= mu1_2;

GaussianBlur(I2_2, sigma2_2, Size(11, 11), 1.5);
sigma2_2 -= mu2_2;

GaussianBlur(I1_I2, sigma12, Size(11, 11), 1.5);
sigma12 -= mu1_mu2;

/////////////////////////////////// FORMULA ////////////////////////////////
Mat t1, t2, t3;

t1 = 2 * mu1_mu2 + C1;
t2 = 2 * sigma12 + C2;
t3 = t1.mul(t2);      // t3 = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))

t1 = mu1_2 + mu2_2 + C1;
t2 = sigma1_2 + sigma2_2 + C2;
t1 = t1.mul(t2);      // t1 =((mu1_2 + mu2_2 + C1).*(sigma1_2 + sigma2_2 + C2))

Mat ssim_map;
divide(t3, t1, ssim_map);        // ssim_map =  t3./t1;

Scalar mssim = mean( ssim_map ); // mssim = average of ssim map
return mssim;
```

```
}
```

Kode 3.16. Validasi SSIM

III.3.4. Pengujian

Pengujian merupakan langkah penelitian yang dilakukan setelah mengerjakan *code project*. Pengujian pada penelitian ini dilakukan dengan melakukan uji coba terhadap seluruh citra yang diuji dan mencatat data waktu komputasi serta mencatat hasil validasi citra yang kemudian digunakan untuk perbandingan untuk mengetahui kelebihan atau kekurangan dari penelitian yang dilakukan.

III.3.5. Analisis

Analisis merupakan langkah yang dilakukan setelah proses pengujian. Analisis yang dilakukan pada penelitian ini adalah membandingkan waktu komputasi antara CPU dan GPU, membandingkan hasil validasi dari citra berdasarkan *masking* yang berbeda, dan membandingkan *speed-up* waktu komputasi dari seluruh citra bahan uji.