BAB V

KESIMPULAN DAN SARAN

Berdasarkan hasil penelitian yang dilakukan dan hasil visualisasi model perambatan gelombang tsunami di perairan laut Aceh dapat diambil beberapa kesimpulan dan saran.

5.1 Kesimpulan

1. Metode *Lattice Boltzmann* dapat digunakan untuk melakukan simulasi fluida.
2. Metode *Lattice Boltzmann* menggunakan asumsi air dangkal (*shallow water*) mampu memberikan hasil visualisasi yang baik.
3. Visualisasi model perambatan gelombang tsunami menggunakan metode *Lattice Boltzmann* memberikan hasil yang baik untuk perambatan gelombang tsunami di perairan Aceh dalam bentuk dua dimensi.

5.2 Saran

1. Penelitian dapat dilakukan pada daerah yang memiliki perairan laut yang dapat terjadi gelombang tsunami.
2. Penelitian ini masih bisa dikembangkan menggunakan arsitektur CUDA-GPU sehingga apabila menggunakan dimensi citra dalam ukuran yang besar maka beban pemrosesan bisa dilakukan oleh *Graphic Processing Unit* (GPU).

3. Untuk penelitian selanjutnya, dapat dilakukan perbandingan dengan satu metode lain yang digunakan untuk simulasi perambatan gelombang tsunami.

DAFTAR PUSTAKA

Adanhounme, Villevo., Codo, Francois de Paule., Admou, Alain., 2012, *Solving the Navier Stokes Flow Equations of Micro-Polar Fluids by Adomian Decomposition Method*. Bulletin of Society for Mathematical Services and Standards, Vol.1 No.2, pp.35-42.

Ancey, C., Iverson, R.M., Rentschler, M., Denlinger, R.P., 2006, *An Exact Solution for Ideal Dam-Break Floods on Steep Slopes*, Water Resources Research, Vol. 44.

Badan Meteorologi Klimatologi dan Geofisika (BMKG) , 2010, *Indonesia Tsunami Early Warning System* (*InaTEWS*), *Konsep dan Implementasi*. Jakarta

Basuki, Achmad., Ramadijanti, Nana., 2006, *Grafika Komputer Teori dan Implementasi*, Penerbit ANDI, Yogyakarta.

Ditya, Didit., 2010, *Tsunami Simulation in Indonesia's Areas Based On Shallow Water Equations And Variational Boussinesq Model Using Finite Element Method*, Thesis Master of Science in the Institut Teknologi Bandung.

George, David L., LeVeque, Randall J, 2006, *Finite Volume Methods And Adaptive Refinement For Global Tsunami Propagation And Local Inundation*, Science of Tsunami Hazards, Vol. 24, No. 5.

Geveler, Markus., Ribbrock, Dirk., Goddeke, Dominik. & Turek, Stefan. 2010. *Lattice- Boltzmann Simulations of the Shallow-Water Equations with Fluid-Structure Interaction on Multi- and Manycore Processor*. Institut fiir Angewandte Mathematik, TU Dortmund, Germany.

Hooper, Daire., Coughlan, Joseph., Mullen, Michael R., 2008, *Stuctural Equation Modelling: Guidelines for Determining Model Fit*, Electronic Journal of Business Research Methods Volume 6 Issue 1 2008 (53-60).

Huang, Chieh-Ling., 2009, *Shape-Based Level Set Method for Image Segmentation*, Ninth International Conference on Hybrid Intelligent Systems.

Ilyas, Tommy., 2006, *Mitigasi Gempa dan Tsunami Didaerah Perkotaan*. Seminar Bidang Kerekayasaan Fakultas Elektro-Unsrat.

Kakiay, T.J., 2004, *Pengantar Sistem Simulasi*. Penerbit Andi Yogyakarta.

Kusuma, M.S.B., Adityawan, M.B., Farid, M., 2008, *Modeling Two Dimension Inundation Flow Generated By Tsunami Propagation In Banda Aceh City*, International Conference on Earthquake Engineering and Disaster Mitigation, Jakarta.

Marghany, Maged., 2012, *Finite Element Method for Simulation of Tsunami Run-up From QuikiBird Satellite Data*, International Journal of Physical Science Vol. 7(9).

Mohamad, A.A., 2011, *Lattice Boltzmann Method: Fundamental and Engineering Applications With Computer Codes*, Springer London Dordrecht Heidelberg New York.

Moriyama, Koji., Inamuro, Takaji., 2011, *Lattice Boltzmann Simulations of Water Transport from the Gas Diffusion Layer to the Gas Channel in PEFC*, Commun. Comput. Phys.

Okatariadi, Oki., 2009, *Peran Kapasitas Bentang Alam Dalam Upaya Kesiapsiagaan Menghadapi Bencana Tsunami Wilayah Pesisir Sukabumi, Jawa Barat*. Buletin Geologi Tata Lingkungan (Bulletin of Environmetal Geology), Vol. 19, No. 1, April 2009:39-49.

Pranowo, 2011, *Pemodelan Numeris Perambatan Gelombang Ultrasonik Berbasis Metode Discontinous Galerkin*, Penerbit Universitas Atma Jaya Yogyakarta, Yogyakarta.

R, Muthukrishnan., Radha, M., 2011, *Edge Detection Techniques for Image Segmentation*, International Journal of Computer Science & Information Technology (IJCSIT) Vol 3, No. 6, Dec 2011.

Ramya, V., Palaniappan, B., 2011, *An Automated Tsunmai Alert System*, International Journal of Embedded System Applications (IJESA) Vol. 1, No. 2, December 2011.

Refrizon., Suwarsono., 2006, *Hubungan Aktivitas Gempa Tektonik Daerah Subduction Indo-Australia Eurasia Segmen Enggano Tahun 2000 Dengan Aktivitas Gempa Vulkanik Gunungapi Kaba dan Dempo.* Jurnal Gradien Vol. 2 No. 2 Juli 2006: 167-171.

Setyonegoro, Wiko., 2009, *Tsunami Numerical Simulation Applied To Tsunami Early Warning System Along Sumatra Region*. JICA Training Course.

Suryani, Erma., 2006, *Pemodelan dan Simulasi*, Graha Ilmu, Yogyakarta.

Thurey, Nills., Rude, Ulrich., Stamminger, Marc., 2006, *Animation of Open Water Phenomena with Coupled Shallow Water and Free Surface Simulations.* Eurographics/ACM SIGGRAPH Symposium on Computer Animation.

Tripathi, D., Chaube, M.K., Gupta, P.K., 2011, *Stokes Flow of Micro-Polar Fluids by Peristaltic Pumping Through Tube with Slip Boundary Condition*. Applied Mathematics and Mechanics.

Tubbs, Kevin., 2010, *Lattice Boltzmann Modelling for Shallow Water Equations Using High Performance Computing*, Program in Engineering Science, Lousiana State University.

Viggen, Erlend Magnus., 2009, *The Lattice Boltzmann Method With Applications in Accourtic*. Thesis Department of Physics – NTNU.

Zhang, Xinming., 2011, *Lattice Boltzmann Implementation for Fluids Flow Simulation in Porous Media.* International Journal Image, Graphics and Signal Processing, 2011, 4, 39-45.

Zimmerman, Mark., 2008, *Modeling Two Dimensional Incompressible Fluid Flow with the Navier Stokes Equations*, Physics Department, The College of Wooster, Wooster, Ohio USA.

LAMPIRAN

Lampiran A : Source Code 2D Lattice Boltzmann Shallow WaterEquation

```
////////////////////////////////////////////////////////////////////////////
//
// Crude 2D Lattice Boltzmann Demo program
// C version
// Graham Pullan - Oct 2008
//
//      f6  f2   f5
//        \  |  /
//         \ | /
//          \|/
//      f3---|--- f1
//          /|\
//         / | \       and f0 for the rest (zero) velocity
//        /  |  \
//      f7  f4   f8
//
////////////////////////////////////////////////////////////////////////////
/

//#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "GL/glew.h"
#include "GL/glut.h"
#include "GL/glu.h"
#include "GL/gl.h"

#define I2D(ni,i,j) (((ni)*(j)) + i)

////////////////////////////////////////////////////////////////////////////

// OpenGL pixel buffer object and texture //
GLuint gl_PBO, gl_Tex;


// arrays //
float *f0,*f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8,*h_surf;
float *tmpf0,*tmpf1,*tmpf2,*tmpf3,*tmpf4,*tmpf5,*tmpf6,*tmpf7,*tmpf8;
float *cmap,*plotvar. *peta;
int *solid;
unsigned int *cmap_rgba, *plot_rgba, *peta_rgba;  //rgba arrays for plotting

// scalars //
float tau,faceq1,faceq2,faceq3,gr;
float vxin, hout, hmin,hmax;
float width, height;
int ni,nj, i0;
int ncol, nrow, iter;
int ipos_old,jpos_old, draw_solid_flag;
```

```
////////////////////////////////////////////////////////////////////////////////

//
// OpenGL function prototypes
//
void display(void);
void resize(int w, int h);
void mouse(int button, int state, int x, int y);
void mouse_motion(int x, int y);
//void shutdown(void);

//
// Lattice Boltzmann function prototypes
//
void stream(void);
void collide(void);
void solid_BC(void);
void per_BC(void);
void in_BC(void);
void ex_BC_crude(void);
void apply_BCs(void);
float h_max(float *h_surf);
float h_min(float *h_surf);

unsigned int get_col(float min, float max, float val);


////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void stream(void)

// Move the f values one grid spacing in the directions that they are pointing
// i.e. f1 is copied one location to the right, etc.

{
    int i,j,im1,ip1,jm1,jp1,i0;

    // Initially the f's are moved to temporary arrays
    for (j=0; j<nj; j++) {
      jm1=j-1;
      jp1=j+1;
      if (j==0) jm1=0;
      if (j==(nj-1)) jp1=nj-1;
      for (i=1; i<ni; i++) {
          i0  = I2D(ni,i,j);
          im1 = i-1;
          ip1 = i+1;
          if (i==0) im1=0;
          if (i==(ni-1)) ip1=ni-1;
          tmpf1[i0] = f1[I2D(ni,im1,j)];
          tmpf2[i0] = f2[I2D(ni,i,jm1)];
          tmpf3[i0] = f3[I2D(ni,ip1,j)];
          tmpf4[i0] = f4[I2D(ni,i,jp1)];
          tmpf5[i0] = f5[I2D(ni,im1,jm1)];
          tmpf6[i0] = f6[I2D(ni,ip1,jm1)];
```

```
            tmpf7[i0] = f7[I2D(ni,ip1,jp1)];
            tmpf8[i0] = f8[I2D(ni,im1,jp1)];
        }
    }

    // Now the temporary arrays are copied to the main f arrays
    for (j=0; j<nj; j++) {
      for (i=1; i<ni; i++) {
            i0 = I2D(ni,i,j);
            f1[i0] = tmpf1[i0];
            f2[i0] = tmpf2[i0];
            f3[i0] = tmpf3[i0];
            f4[i0] = tmpf4[i0];
            f5[i0] = tmpf5[i0];
            f6[i0] = tmpf6[i0];
            f7[i0] = tmpf7[i0];
            f8[i0] = tmpf8[i0];
      }
    }
}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void collide(void)

// Collisions between the particles are modeled here. We use the very simplest
// model which assumes the f's change toward the local equlibrium value (based
// on density and velocity at that point) over a fixed timescale, tau

{
    int i,j,i0;
    float h, hvx, hvy, vx, vy, v_sq_term;
    float f0eq, f1eq, f2eq, f3eq, f4eq, f5eq, f6eq, f7eq, f8eq;
    float rtau, rtau1;


    // Some useful constants
    rtau = 1.f/tau;
    rtau1 = 1.f - rtau;

    for (j=0; j<nj; j++) {
      for (i=0; i<ni; i++) {

            i0 = I2D(ni,i,j);

      // Do the summations needed to evaluate the density and components of
         velocity
      h =  f0[i0] + f1[i0] + f2[i0] + f3[i0] + f4[i0] + f5[i0] + f6[i0] +
           f7[i0] + f8[i0];
             h_surf[i0]=h;

            hvx = f1[i0] - f3[i0] + f5[i0] - f6[i0] - f7[i0] + f8[i0];
            hvy = f2[i0] - f4[i0] + f5[i0] + f6[i0] - f7[i0] - f8[i0];
            vx = hvx/h;
            vy = hvy/h;
```

```
            // Also load the velocity magnitude into plotvar - this is what we
               will
            // display using OpenGL later
            plotvar[i0] = h;//sqrt(vx*vx + vy*vy);

            v_sq_term = (vx*vx + vy*vy);

            // Evaluate the local equilibrium f values in all directions
              f0eq = h*(1.0 - 5.0*gr*h/6-2.0*v_sq_term/3.0);
              f1eq = h*(gr*h/6.0+vx/3.0 + vx*vx/2.0 - v_sq_term/6.0);
              f2eq = h*(gr*h/6.0+vy/3.0 + vy*vy/2.0 - v_sq_term/6.0);
              f3eq = h*(gr*h/6.0-vx/3.0 + vx*vx/2.0 - v_sq_term/6.0);
              f4eq = h*(gr*h/6.0-vy/3.0 + vy*vy/2.0 - v_sq_term/6.0);
              f5eq = h*(gr*h/6.0+( vx + vy)/3.0 + ( vx + vy)*( vx + vy)/2.0 -
                     v_sq_term/6.0)/4.0;
              f6eq = h*(gr*h/6.0+(-vx + vy)/3.0 + (-vx + vy)*(-vx + vy)/2.0 -
                     v_sq_term/6.0)/4.0;
              f7eq = h*(gr*h/6.0+(-vx - vy)/3.0 + (-vx - vy)*(-vx - vy)/2.0 -
                     v_sq_term/6.0)/4.0;
              f8eq = h*(gr*h/6.0+( vx - vy)/3.0 + ( vx - vy)*( vx - vy)/2.0 -
                     v_sq_term/6.0)/4.0;

            // Simulate collisions by "relaxing" toward the local equilibrium
            f0[i0] = rtau1 * f0[i0] + rtau * f0eq;
            f1[i0] = rtau1 * f1[i0] + rtau * f1eq;
            f2[i0] = rtau1 * f2[i0] + rtau * f2eq;
            f3[i0] = rtau1 * f3[i0] + rtau * f3eq;
            f4[i0] = rtau1 * f4[i0] + rtau * f4eq;
            f5[i0] = rtau1 * f5[i0] + rtau * f5eq;
            f6[i0] = rtau1 * f6[i0] + rtau * f6eq;
            f7[i0] = rtau1 * f7[i0] + rtau * f7eq;
            f8[i0] = rtau1 * f8[i0] + rtau * f8eq;
        }
    }
            hmax=h_max(h_surf);
           hmin=h_min(h_surf);

}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void solid_BC(void)

// This is the boundary condition for a solid node. All the f's are reversed -
// this is known as "bounce-back"

{
    int i,j,i0;
    float f1old,f2old,f3old,f4old,f5old,f6old,f7old,f8old;

    for (j=0;j<nj;j++){
      for (i=0;i<ni;i++){
          i0=I2D(ni,i,j);
          if (solid[i0]==0) {
            f1old = f1[i0];
```

```
            f2old = f2[i0];
            f3old = f3[i0];
            f4old = f4[i0];
            f5old = f5[i0];
            f6old = f6[i0];
            f7old = f7[i0];
            f8old = f8[i0];

            f1[i0] = f3old;
            f2[i0] = f4old;
            f3[i0] = f1old;
            f4[i0] = f2old;
            f5[i0] = f7old;
            f6[i0] = f8old;
            f7[i0] = f5old;
            f8[i0] = f6old;
          }
      }
    }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void per_BC(void)

// All the f's leaving the bottom of the domain (j=0) enter at the top
  (j=nj-1),
// and vice-verse

{
    int i0,i1,i;

    for (i=0; i<ni; i++){
      i0 = I2D(ni,i,0);
      i1 = I2D(ni,i,nj-1);
      f2[i0] = f2[i1];
      f5[i0] = f5[i1];
      f6[i0] = f6[i1];
      f4[i1] = f4[i0];
      f7[i1] = f7[i0];
      f8[i1] = f8[i0];
    }
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void in_BC(void)

// This inlet BC is extremely crude but is very stable
// We set the incoming f values to the equilibirum values assuming:
// h=hout; vx=vxin; vy=0

{
    int i0, j;
    float f1new, f5new, f8new, vx_term;
```

```
    vx_term = 1.f + 3.f*vxin +3.f*vxin*vxin;
    f1new = hout * faceq2 * vx_term;
    f5new = hout * faceq3 * vx_term;
    f8new = f5new;

//    printf("\n vxin = %f",vxin);
    for (j=0; j<nj; j++){
      i0 = I2D(ni,0,j);
      f1[i0] = f1new;
      f5[i0] = f5new;
      f8[i0] = f8new;
    }

}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void ex_BC_crude(void)

// This is the very simplest (and crudest) exit BC. All the f values pointing
// into the domain at the exit (ni-1) are set equal to those one node into
// the domain (ni-2)

{
    int i, i0, i1, j;



      // left side
      for (j=0; j<nj; j++){
      i0 = I2D(ni,0,j);
      i1 =  i0+1;
      f1[i0] = f1[i1];
      f5[i0] = f5[i1];
      f8[i0] = f8[i1];
    }

      // right side;
      for (j=0; j<nj; j++){
      i0 = I2D(ni,ni-1,j);
      i1 = i0 - 1;
      f3[i0] = f3[i1];
      f6[i0] = f6[i1];
      f7[i0] = f7[i1];
    }

      // bottom side
      for (i=0; i<ni; i++){
      i0 = I2D(ni,i,0);
      i1 = I2D(ni,i,1);
      f2[i0] = f2[i1];
      f5[i0] = f5[i1];
      f6[i0] = f6[i1];
      }
```

```
      // top side
      for (i=0; i<ni; i++){
      i0 = I2D(ni,i,nj-1);
      i1 = I2D(ni,i,nj-2);

      f4[i0] = f4[i1];
      f7[i0] = f7[i1];
      f8[i0] = f8[i1];

    }

}


////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

void apply_BCs(void)

// Just calls the individual BC functions

{
  //  per_BC();

    solid_BC();

   // in_BC();

    ex_BC_crude();
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void display(void)

// This function is called automatically, over and over again,  by GLUT

{
    int i,j,ip1,jp1,i0,icol,i1,i2,i3,i4,isol;
    float minvar,maxvar,frac;
     iter+=1;

    // set upper and lower limits for plotting
    minvar=hmin;
    maxvar=hmax;

    // do one Lattice Boltzmann step: stream, BC, collide:
    stream();
    apply_BCs();
    collide();

    // convert the plotvar array into an array of colors to plot
    // if the mesh point is solid, make it black
    for (j=0;j<nj;j++){
      for (i=0;i<ni;i++){
```

```
            i0=I2D(ni,i,j);
            frac=(plotvar[i0]-minvar)/(maxvar-minvar);
            icol=frac*ncol;
            isol=(int)solid[i0];
            plot_rgba[i0] = isol*cmap_rgba[icol];
        }
    }

    // Fill the pixel buffer with the plot_rgba array
    glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,ni*nj*sizeof(unsigned int),
             (void **)plot_rgba,GL_STREAM_COPY);

    // Copy the pixel buffer to the texture, ready to display
    glTexSubImage2D(GL_TEXTURE_2D,0,0,0,ni,nj,GL_RGBA,GL_UNSIGNED_BYTE,0);

    // Render one quad to the screen and colour it using our texture
    // i.e. plot our plotvar data to the screen
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);



      glTexCoord2f (0.0, 0.0);
      glVertex3f (0.0, 0.0, 0.0);
      glTexCoord2f (0.0, 1.0);
      glVertex3f (ni, 0.0, 0.0);

      ////////////////////////////////

      glTexCoord2f (1.0, 1.0);
      glVertex3f (ni, nj, 0.0);
      glTexCoord2f (0.0, 1.0);
      glVertex3f (0.0, nj, 0.0);

      glEnd();
      glutSwapBuffers();

      iter+=1;
            if (iter%1==0)
            {
                  printf(" iterasi = %4d  ; t= %4d \n", iter);
            }

            if (iter==15)
            {
                  system("PAUSE");
                  exit(0);
            }}
}

////////////////////////////////////////////////////////////////////////////////


float h_max(float *h_surf)
{
      int i,totpoints;
      float hmax =-1.0e6;
```

```
        totpoints=ni*nj;
    for ( i=0; i<totpoints-1; i++) {
          if (h_surf[i] > hmax)
                hmax=h_surf[i];
    }
    return hmax;
}

float h_min(float *h_surf)
{
       int i,totpoints;
       float hmin =1.0e6;
        totpoints=ni*nj;
    for ( i=0; i<totpoints-1; i++) {
          if (h_surf[i] < hmin)
                hmin=h_surf[i];
    }
    return hmin;
}

////////////////////////////////////////////////////////////////////////

void resize(int w, int h)

// GLUT resize callback to allow us to change the window size

{
    width = w;
    height = h;
    glViewport (0, 0, w, h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho (0., ni, 0., nj, -200. ,200.);
    glMatrixMode (GL_MODELVIEW);
    glLoadIdentity ();
}


////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////

void mouse(int button, int state, int x, int y)

// GLUT mouse callback. Left button draws the solid, right button removes
solid

{
     float xx,yy;

     if ((button == GLUT_LEFT_BUTTON) && (state == GLUT_DOWN)) {
          draw_solid_flag = 0;
          xx=x;
          yy=y;
          ipos_old=xx/width*ni;
          jpos_old=(height-yy)/height*nj;
     }
```

```
    if ((button == GLUT_RIGHT_BUTTON) && (state == GLUT_DOWN)) {
        draw_solid_flag = 1;
        xx=x;
        yy=y;
        ipos_old=xx/width*ni;
        jpos_old=(height-yy)/height*nj;
    }
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

void mouse_motion(int x, int y)

// GLUT call back for when the mouse is moving
// This sets the solid array to draw_solid_flag as set in the mouse callback
// It will draw a staircase line if we move more than one pixel since the
// last callback - that makes the coding a bit cumbersome:

{
    float xx,yy,frac;
    int ipos,jpos,i,j,i1,i2,j1,j2, jlast, jnext;
    xx=x;
    yy=y;
    ipos=(int)(xx/width*(float)ni);
    jpos=(int)((height-yy)/height*(float)nj);

    if (ipos <= ipos_old){
        i1 = ipos;
        i2 = ipos_old;
        j1 = jpos;
        j2 = jpos_old;
    }
    else {
        i1 = ipos_old;
        i2 = ipos;
        j1 = jpos_old;
        j2 = jpos;
    }

    jlast=j1;

    for (i=i1;i<=i2;i++){
        if (i1 != i2) {
            frac=(float)(i-i1)/(float)(i2-i1);
            jnext=(int)(frac*(j2-j1))+j1;
        }
        else {
            jnext=j2;
        }
        if (jnext >= jlast) {
            solid[I2D(ni,i,jlast)]=draw_solid_flag;
            for (j=jlast; j<=jnext; j++){
                solid[I2D(ni,i,j)]=draw_solid_flag;
            }
        }
        else {
```

```
            solid[I2D(ni,i,jlast)]=draw_solid_flag;
            for (j=jnext; j<=jlast; j++){
                solid[I2D(ni,i,j)]=draw_solid_flag;
            }
        }
        jlast = jnext;
    }


    ipos_old=ipos;
    jpos_old=jpos;
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

int main(int argc, char **argv)
{
    int array_size_2d,totpoints,i,j,i0,i1, A[400][400];
    float rcol,gcol,bcol;

    FILE *fp_col;

    // The following parameters are usually read from a file, but
    // hard code them for the demo:


      ni=400;   // arah mendatar (horizontal)
      nj=400;   // arah tegak (vertikal)
      //vxin=0.0;
      hout=1.0;
      //tau=1.0;
      tau=0.51;
      gr=0.5;
      iter=0;

    // End of parameter list

    // Write parameters to screen
    printf ("ni = %d\n", ni);
    printf ("nj = %d\n", nj);
    printf ("vxin = %f\n", vxin);
    printf ("hout = %f\n", hout);
    printf ("tau = %f\n", tau);
    printf ("gr = %f\n", gr);


    totpoints=ni*nj;
    array_size_2d=ni*nj*sizeof(float);

    // Allocate memory for arrays

    f0 = malloc(array_size_2d);
    f1 = malloc(array_size_2d);
    f2 = malloc(array_size_2d);
    f3 = malloc(array_size_2d);
    f4 = malloc(array_size_2d);
```

```
 f5 = malloc(array_size_2d);
 f6 = malloc(array_size_2d);
 f7 = malloc(array_size_2d);
 f8 = malloc(array_size_2d);
   h_surf = malloc(array_size_2d);

 tmpf0 = malloc(array_size_2d);
 tmpf1 = malloc(array_size_2d);
 tmpf2 = malloc(array_size_2d);
 tmpf3 = malloc(array_size_2d);
 tmpf4 = malloc(array_size_2d);
 tmpf5 = malloc(array_size_2d);
 tmpf6 = malloc(array_size_2d);
 tmpf7 = malloc(array_size_2d);
 tmpf8 = malloc(array_size_2d);

 plotvar = malloc(array_size_2d);

 plot_rgba = malloc(ni*nj*sizeof(unsigned int));

 solid = malloc(ni*nj*sizeof(int));

 //
 // Some factors used to calculate the f_equilibrium values
 //
 faceq1 = 4.f/9.f;
 faceq2 = 1.f/9.f;
 faceq3 = 1.f/36.f;


 //
 // Initialise f's by setting them to the f_equilibirum values assuming
 // that the whole domain is at velocity vx=vxin vy=0 and density h=hout
 //



 for ( j=0; j<nj; j++) {
   for ( i=0; i<ni; i++) {
         int  i0 = I2D(ni,i,j);
         h_surf[i0] =1.0+0.2*expf(-0.050*((1.0*i-150.0)*(1.0*i-
                    150.0)+(1.0*j-150.0)*(1.0*j-150.0)));
   }
}


 for (i=0; i<totpoints; i++) {
   f0[i] = h_surf[i]*(1.0 - 5.0*gr*h_surf[i]/6-
           2.0*vxin*vxin/3.0);
   f1[i] = h_surf[i]*(gr*h_surf[i]/6.0 + vxin/3.0 + vxin*vxin/2.0 -
           vxin*vxin/6.0);
   f2[i] = h_surf[i]*(gr*h_surf[i]/6.0 +  0.0/3.0 + 0.0*0.0/2.0   - \
           vxin*vxin/6.0);
   f3[i] = h_surf[i]*(gr*h_surf[i]/6.0 - vxin/3.0 + vxin*vxin/2.0 -
           vxin*vxin/6.0);
   f4[i] = h_surf[i]*(gr*h_surf[i]/6.0 -  0.0/3.0 +   0.0*0.0/2.0 -
           vxin*vxin/6.0);
```

```
  f5[i] = h_surf[i]*(gr*h_surf[i]/6.0+( vxin + 0.0)/3.0 + ( vxin + 0.0)*(
          vxin + 0.0)/2.0 - vxin*vxin/6.0)/4.0;
  f6[i] = h_surf[i]*(gr*h_surf[i]/6.0+(-vxin + 0.0)/3.0 + (-vxin + 0.0)*(-
          vxin + 0.0)/2.0 - vxin*vxin/6.0)/4.0;
  f7[i] = h_surf[i]*(gr*h_surf[i]/6.0+(-vxin - 0.0)/3.0 + (-vxin - 0.0)*(-
          vxin - 0.0)/2.0 - vxin*vxin/6.0)/4.0;
  f8[i] = h_surf[i]*(gr*h_surf[i]/6.0+( vxin - 0.0)/3.0 + ( vxin - 0.0)*(
          vxin - 0.0)/2.0 - vxin*vxin/6.0)/4.0;
  plotvar[i] = h_surf[i];
  solid[i] = 1;
}

   hmin=1.0;
   hmax=1.2;

fp_col = fopen("map400.dat","r");
if (fp_col==NULL) {
  printf("Error: can't open file peta \n");
  return 1;
}
// allocate memory for colourmap (stored as a linear array of int's)
fscanf (fp_col, "%d", &ncol);
  fscanf (fp_col, "%d", &nrow);

peta_rgba = (unsigned int *)malloc(ncol*sizeof(unsigned int));
peta_rgba = (unsigned int *)malloc(nrow*sizeof(unsigned int));

  // read colourmap and store as int's

for (i=0;i<ncol;i++){
        for (j=0;j<nrow;j++){
              fscanf(fp_col, "%f", &rcol);//, &gcol, &bcol);
              A[i][j]=rcol;
              if(rcol == 1){
                    i0=I2D(ni,i,j);
                    solid[i0]=1;
              }
              else{
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
              }
        }
}


//
// Read in colourmap data for OpenGL display
//
  fp_col = fopen("cmap.dat","r");
if (fp_col==NULL) {
  printf("Error: can't open cmap.dat \n");
  return 1;
}
// allocate memory for colourmap (stored as a linear array of int's)
fscanf (fp_col, "%d", &ncol);
cmap_rgba = (unsigned int *)malloc(ncol*sizeof(unsigned int));
// read colourmap and store as int's
```

```
for (i=0;i<ncol;i++){
  fscanf(fp_col, "%f%f%f", &rcol, &gcol, &bcol);
  cmap_rgba[i]=((int)(255.0f) << 24) | // convert colourmap to int
      ((int)(bcol * 255.0f) << 16) |
      ((int)(gcol * 255.0f) <<  8) |
      ((int)(rcol * 255.0f) <<  0);
}
fclose(fp_col);


//
// Iinitialise OpenGL display - use glut
//
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(ni, nj);      // Window of ni x nj pixels
glutInitWindowPosition(50, 50);  // position
glutCreateWindow("2D LB");       // title

// Check for OpenGL extension support
printf("Loading extensions: %s\n", glewGetErrorString(glewInit()));
if(!glewIsSupported(
                  "GL_VERSION_2_0 "
                  "GL_ARB_pixel_buffer_object "
                  "GL_EXT_framebuffer_object "
                  )){
    fprintf(stderr, "ERROR: Support for necessary OpenGL extensions
    missing.");
    fflush(stderr);
    return;
}

// Set up view
glClearColor(0.0, 0.0, 0.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0,ni,0.,nj, -200.0, 200.0);


// Create texture which we use to display the result and bind to gl_Tex
glEnable(GL_TEXTURE_2D);
glGenTextures(1, &gl_Tex);                    // Generate 2D texture
glBindTexture(GL_TEXTURE_2D, gl_Tex);         // bind to gl_Tex
// texture properties:
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, ni, nj, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, NULL);


// Create pixel buffer object and bind to gl_PBO. We store the data we
   want to
// plot in memory on the graphics card - in a "pixel buffer". We can then
// copy this to the texture defined above and send it to the screen
glGenBuffers(1, &gl_PBO);
```

```
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, gl_PBO);
    printf("Buffer created.\n");


    // Set the call-back functions and start the glut loop
    printf("Starting GLUT main loop...\n");
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutIdleFunc(display);
    glutMouseFunc(mouse);
    glutMotionFunc(mouse_motion);
    glutMainLoop();

    return 0;
}
```

Lampiran B : Source Code Memunculkan Citra RGB (MATLAB)

```
I=imread('400x400.png');
Ir=double(I(:,:,1));
Ir=flipdim(Ir ,1); % vertical flip
surf(Ir); shading interp;
%imshow(I);
imwrite(Ir, 'E:\PROGRAM TESIS\LevelSet\peta.png', 'png');
clear I;
```

Lampiran C : Source Code Konversi Citra Warna Ke Citra Biner (MATLAB)

```
I=imread('400x400.png');
Ir=double(I(:,:,1));

[imax,jmax]=size(Ir);

A=double(Ir);
B=double(Ir);
for i=1:imax
    for j=1:jmax
        if Ir(i,j)<=200
            A(i,j)=0;
        else

            A(i,j)=1;
        end
    end
end
Ir=flipdim(Ir ,1); % vertical flip
surf(Ir); shading interp;
imshow(A);
imwrite(A, 'E:\PROGRAM TESIS\visualisasi\map_aceh\400px400p_bw.png', 'png');
clear I;
```

Lampiran D : Source Code Merubah Citra Menjadi Citra Biner (MATLAB)

```
clc;
clear all;
close all;
I=imread('400x400_bw.png');
figure,imshow(I)
%% menyimpan hasil matrix ke format ASCII
BW=double(I);

save('map400.dat', 'BW', '-ASCII');
```

DAFTAR RIWAYAT HIDUP

Curriculum Vitae

I. KETERANGAN PRIBADI

1	Nama Lengkap (sesuai Ijasah)		Nazaruddin Ahmad
2	Tempat Lahir/Tgl. Lahir (sesuai Ijazah)		Banda Aceh/05 Juni 1982
3.	Jenis Kelamin		a. Pria
4.	Agama		Islam
5.	Alamat Rumah (sesuai KTP)	a. Jalan	Makam T. Nyak Arief No. 09
		b. Kelurahan/Desa	Meunasah Papeun
		c. Kecamatan	Krueng Barona Jaya
		d. Kabupaten/Kota	Aceh Besar
		e. Propinsi/ kode pos	Nanggroe Aceh Darussalam / 23373
		f. Telp rumah	0651- 7552655
6.	Alamat Surat	a. Jalan	Makam T. Nyak Arief No. 09
		b. Kelurahan/Desa	Meunasah Papeun
		c. Kecamatan	Krueng Barona Jaya
		d. Kabupaten/Kota	Aceh Besar
		e. Propinsi/ kode pos	Nanggroe Aceh Darussalam / 23373
		f. Telp / HP	0651-7552655 / 081360866064
7.	Kegemaran (Hobby)		Membaca
8.	Alamat e-mail		nazar.ahmadhb@gmail.com nazar.lbs@gmail.com
9.	Surat Ijin Mengemudi (SIM)		SIM A dan SIM C
10.	Bahasa Asing yang dikuasai (Aktif/ Pasif)		Bahasa Inggris (Pasif)
11.	Keahlian Bidang Komputer		- Bahasa Pemrograman VB - Database (MYSQL,SQL) - Bahasa C++

II. P E N D I D I K A N

1. Pendidikan

NO.	TINGKAT	NAMA PEN DIDIKAN	JURUSAN	STTB/ TANDA LULUS/ IJAZAH TAHUN	TEMPAT
1	2	3	4	5	6
1	SD	MIN 1	-	1994	Banda Aceh
2	SLTP	MTsN 1	-	1997	Banda Aceh
3	SLTA/ Sederajat	SMU 3	IPS/Sosial	2000	Banda Aceh
4	D-III	-	-	-	-
5.	S 1	Univ. Jabal Ghafur Sigli	Teknik Informatika	2008	Banda Aceh
6.	S 2	Univ. Atma Jaya Yogyakarta	Magister Teknik Informatika	2013	Yogyakarta

2. Kursus/ Latihan di Dalam dan di Luar Negeri

NO.	NAMA KURSUS/ LATIHAN	LAMANYA (TGL/BLN/THN s.d. TGL/BLN/THN)	IJAZAH/TANDA LULUS/SURAT KETERANGAN TAHUN	TEMPAT	KETERANGAN
1	2	3	4	5	6

III. RIWAYAT PEKERJAAN

NO.	NAMA INSTANSI	JABATAN	Mulai bekerja (tgl)	Berhenti (tgl)
1	2	3		4
1.	PT. USMB Banda Aceh	Staff Administrasi	03-03-2003	03-06-2006
2.	Univ. Jabal Gahfur Sigli	Staff Pengajar	20-09-2008	-
3.	STMIK U'Budiyah Banda Aceh	Staff Pengajar	10-03-2009	15-05-2011
4.				

IV. MAKALAH/POSTER

NO.	TAHUN	JUDUL	PENYELENGGARA
1	2	3	
1.	2012	Metode Histogram Equalization Untuk Perbaikan Citra	Seminar Nasional Teknologi Informasi & Komunikasi Terapan 2012 Universitas Dian Nuswantoro Semarang
2.	2012	Metode Lattice Boltzmann Untuk Perambatan Gelombang Air	Seminar Nasional Ilmu Komputer 2012 Universitas Diponegoro Semarang
3.	2013	Model 2D Visualisasi Tsunami Aceh Dengan Metode Lattice Boltzmann	Seminar Nasional Teknologi Informasi dan Komunikasi 2013 Universitas Atma Jaya Yogyakarta

V. KONFERENSI/SEMINAR/LOKAKARYA/SIMPOSIUM

NO.	TAHUN	JUDUL KEGIATAN	PENYELENGGARA	PANITIA/PESERTA/ PEMAKALAH
1	2	3	4	5
1.	2012	SEMANTIK	Universitas Dian Nuswantoro Semarang	PEMAKALAH
2.	2012	SNIK UNDIP	Universitas Diponegoro Semarang	PEMAKALAH
3.	2013	SENTIKA UAJY	Univeristas Atma Jaya Yogyakarta	PEMAKALAH

Demikian Daftar Riwayat Hidup ini saya buat dengan sesungguhnya dan dapat digunakan seba**gaimana mestinya.**

Yang Membuat,

(Nazaruddin Ahmad, S.T, M.T)