BAB 6
KESIMPULAN DAN SARAN

6.1. Kesimpulan

Berdasarkan hasil penelitian Simulasi Gelombang Air dengan Metode Finite Volume Berbasis Komputasi Paralel GPU CUDA, dapat ditarik beberapa kesimpulan, yaitu:

1. Gelombang air yang didasarkan pada model persamaan air dangkal telah berhasil disimulasikan menggunakan metode *finite volume central upwind* pada *mesh* segitiga tidak terstruktur, dengan memanfaatkan CPU maupun GPU CUDA.
2. Penggunaan GPU CUDA, GeForce GTX 660 Ti, pada simulasi gelombang air dengan metode *finite volume*, terbukti dapat mempercepat proses komputasi hingga 8 kali CPU. Adapun peningkatan kecepatan ini dipengaruhi oleh variasi jumlah elemen dan alokasi blok *thread*, yaitu semakin besar jumlah elemen semakin meningkat pula kecepatan optimal yang dapat diraih sebelum mencapai kondisi stagnan, serta alokasi blok *thread* terbaik yaitu sebesar 128 *thread* per blok.
3. Visualisasi grafis gelombang air dalam tiga dimensi berhasil dijalankan secara *real-time*, dengan memanfaatkan GLSL (OpenGL Shader Language) dan fitur VBO (Vertex Buffer Object).

6.2. Saran

Beberapa saran dari penulis untuk penelitian bagi simulasi gelombang air dengan metode *finite volume* secara paralel adalah sebagai berikut.

1. Penelitian ini dapat dikembangkan dengan menambahkan topografi dasar yang tidak rata, serta daerah kering atau daratan.
2. Penelitian ini dapat dikembangkan dengan menggunakan multi GPU, bahkan *clutser* dengan multi GPU.

DAFTAR PUSTAKA

Ahmad, M. F., Mamat, M., Nik, W. B. W. & Kartono, A., 2013. Numerical method for dam break problem by using Godunov approach. *Applied Mathematics and Computational Intelligence,* 2(1), pp. 95-107.

Balbas, J. & Hernandez-Duenas, G., 2014. A positivity preserving central scheme for shallow water flows in channels with wet-dry states. *ESAIM Mathematical Modelling and Numerical Analysis,* 48(3), pp. 665-696.

Banks, J., Carson, J. S., Nelson, B. L. & Nicol, D. M., 2004. *Discrete-Event System Simulation.* 4th ed. s.l.:Prentice Hall.

Benkhaldoun, F. & Mohammed, S., 2010. A simple finite volume method for the shallow water equations. *Journal of Computational and Applied Mathematics,* 234(1), pp. 58-72.

Bern, M. & Plassman, P., 2000. Mesh Generation. In: J. -. Sack & J. Urrutia, eds. *Handbook of Computational Geometry.* Amsterdam: Elsevier Science, pp. 291-315.

Bloom, C., 2011. *ABC News.* [Online] Tersedia di: http://www.abc.net.au/news/2011-01-10/water-spills-over-the-burdekin-dam-wall/1889116 [Diakses 4 Juli 2016].

BNPB, 2016. *Statistik - Data Bencana: Data dan Informasi Bencana Indonesia, Badan Nasional Penanggulangan Bencana.* [Online] Tersedia di: http://dibi.bnpb.go.id/data-bencana/statistik [Diakses 8 Juli 2016].

Bollermann, A., Chen, G., Kurganov, A. & Noelle, S., 2013. A Well-Balanced Reconstruction of Wet/Dry Fronts for The Shallow Water Equations. *Journal of Scientific Computing,* 56(2), pp. 267-290.

Brodtkorb, A. R., Saetra, M. L. & Altinakar, M., 2012. Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation. *Computers & Fluids,* 55(1), pp. 1-12.

Bryson, S., Epshteyn, Y., Kurganov, A. & Petrova, G., 2011. *Central-Upwind Scheme on Triangular Grids for the Saint-Venant System of Shallow Water Equation.* Halkidiki, (Greece), American Institute of Physics.

Bryson, S., Epshteyn, Y., Kurganov, A. & Petrova, G., 2011. Well-balanced positivity preserving central-upwind scheme on triangular grids for the Saint-Venant system. *ESAIM: Mathematical Modelling and Numerical Analysis,* 45(3), pp. 423-446.

Burnes, A., 2012. *Meet Your New Weapon: The GeForce GTX 660 Ti. Borderlands 2 Included..* [Online]
Tersedia di: http://www.geforce.com/whats-new/articles/geforce-gtx-660-ti-borderlands-2
[Diakses 25 Juli 2016].

Chapra, S. C. & Canale, R. P., 2015. *Numerical Methods for Engineers.* 7th ed. USA: McGraw-Hill Education.

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

Couason, P. D. M., Mingham, P. C. G. & Qian, D. L., 2011. *Introductory Finite Volume Methods for PDEs.* s.l.:Ventus Publishing ApS.

Furgerot, L., Weill, P., Mouaze, D. & Tessier, B., 2016. Suspended sediment dynamics induced by the passsage of a tidal bore in an upper estuary. In: J. Reynaud & B. Tessier, eds. *Contributions to modern and ancient tidal sedimentology : Proceedings of the Tidalites 2012 Conference.* Reynaud, Jean-Yves; Tessier, Bernadette ed. United States: John Wiley & Sons, pp. 61-74.

Griffiths, D. F., Dold, J. W. & Silvester, D. J., 2015. *Essential Partial Differential Equations: Analytical and Computational Aspects.* Switzerland: Springer.

Hidayat, N., Suhariningsih, Suryanto, A. & Mungkasi, S., 2014. The Significance of Spatial Reconstruction in Finite Volume Methods for the Shallow Water Equations. *Applied Mathematical Sciences,* 8(29(2)), pp. 1411-1420.

Hirsch, C., 2007. *Numercial Computation of Internal and External Flows (Volume 1), Fundamentals of Computational Fluid Dynamics.* 2nd ed. Great Britain: Butterworth-Heinemann.

INFORM, 2016. *INFORM country risk profiles for 191 countries.* [Online] Tersedia di: http://www.inform-index.org/Countries/Country-profiles/iso3/IDN
[Diakses 8 Juli 2016].

Kaushik, 2014. *Amusing Planet.* [Online] Tersedia di: http://www.amusingplanet.com/2014/01/tidal-bore-when-rivers-flow-against.html
[Diakses 4 Juli 2016].

Khronos, 2016. *Khronos Logos, Trademarks, and Guidelines.* [Online] Tersedia di: https://www.khronos.org/legal/trademarks//
[Diakses 25 Juli 2016].

Kurganov, A., Noelle, S. & Petrova, G., 2001. Semidiscrete Central-Upwind Schemes for Hyperbolic Conservation Laws and Hamilton--Jacobi Equations. *SIAM Journal on Scientific Computing*, 23(3), pp. 707-740.

Kurganov, A. & Petrova, G., 2005. Central-upwind schemes on triangular grids for hyperbolic systems of conservation laws. *Numerical Methods for Partial Differential Equations,* 21(3), pp. 536-552.

Kurganov, A. & Petrova, G., 2007. A Second-Order Well-Ballanced Positivity Preserving Central-Upwind Scheme for The Saint-Venant System. *Communications in Mathematical Sciences,* 5(1), pp. 133-160.

Lautrup, B., 2011. *Physics of Continuous Matter, Second Edition : Exotic and Everyday Phenomena in the Macroscopic World.* 2nd ed. Hoboken: CRC Press.

LeVeque, R. J., 2002. *Finite Volume Methods for Hyperbolic Problems.* Cambridge: Cambridge University Press.

Matsuyama, M. & Tanaka, H., 2001. *An experimental study of the highest run-up height in the 1993 Hokkaido Nansei-Oki earthquake tsunami.* Seattle, U.S.A., International Tsunami Symposium 2001.

Moukalled, F., Mangani, L. & Darwish, M., 2016. *The Finite Volume in Computational Fluid Dynamics, An Advanced Introduction with OpenFOAM and Matlab.* Switzerland: Springer.

Nielsen, O. M., Roberts, S. G., Gray, D. & Hitchman, A., 2005. *Hydrodynamic modelling of coastal inundation.* s.l., Modelling and Simulation Society of Australia and New Zealand, pp. 518-523.

NOAA, 2015. *MARCH 11, 2011 JAPAN EARTHQUAKE AND TSUNAMI.* [Online] Tersedia di: https://www.ngdc.noaa.gov/hazard/data/publications/2011_0311.pdf [Diakses 8 Juli 2016].

NVIDIA, 2016. *Nsight Eclipse Edition.* [Online] Tersedia di: https://developer.nvidia.com/nsight-eclipse-edition [Diakses 25 Juli 2016].

Officer, C. B., 1974. *Introductin to Theoritical Geophysics.* 1st ed. New York: Springer-Verlag.

Passos, W. D., 2010. *Numerical Methods, Algorithms and Tools in C#.* Boca Raton: CRC Press.

Roberts, S. et al., 2015. *ANUGA User Manual Release 2.0.* s.l.:Commonwealth of Australia (Geoscience Australia) and the Australian National University.

Shearer, M. & Levy, R., 2015. *Partial Differential Equation: An Introduction to Theory and Applications.* New Jersy: Princeton University Press.

Shimbun, Y., 2011. *Japan Tsunami: 20 Unforgettable Pictures.* [Online] Tersedia di: http://news.nationalgeographic.com/news/2011/03/pictures/110315-nuclear-reactor-japan-tsunami-earthquake-world-photos-meltdown/ [Diakses 4 Juli 2016].

Shirkhani, H., Mohammadian, A., Seidou, O. & Kurganov, A., 2016. A well-balanced positivity-preserving central-upwind scheme for shallow water equations on unstructured quadrilateral grids. *Computers & Fluids,* 126(3), pp. 25-40.

Shirkhani, H., Mohammadian, A., Seidou, O. & Qiblawey, H., 2015. Analysis of Triangular C-Grid Finite Volume Scheme for Shallow Water. *Advances in Water Resources,* 82(14), pp. 176-195.

Shirley, P. & Marschner, S., 2009. *Fundamental of Computer Graphics.* 3rd ed. Boca Raton: CRC Press.

Staff, H., 2009. *Yangtze River peaks in China.* [Online] Tersedia di: http://www.history.com/this-day-in-history/yangtze-river-peaks-in-china [Diakses 05 Juli 2016].

Ubuntu, 2016. *Ubuntu Design.* [Online] Tersedia di: http://design.ubuntu.com/brand/ubuntu-logo [Diakses 25 Juli 2016].

Versteeg, H. K. & Malalasekera, W., 2007. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method.* 2nd ed. England: Pearson Education Limited.

Vialar, T., 2015. *Handbook of Mathematics.* Paris: Books on Demands.

Zoppou, C. & Roberts, S., 2000. Numerical solution of the two-dimensional unsteady dam break. *Applied Mathematical Modelling*, 24(7), pp. 457-475.

LAMPIRAN A

Statistik Jumlah Korban dan Kerusakan Bencana Hidrologi

Berikut adalah data jumlah korban dan kerusakan akibat berbagai bencana hidrologi yang terjadi di Indonesia hingga tahun 2016. Bencana yang dimaksud mencakup banjir, tanah longsor, gelombang pasang, serta gempa dan tsunami.

Tabel A.1. Jumlah Korban dan Kerusakan akibat Bencana Hidrologi di Indonesia Hingga Tahun 2016

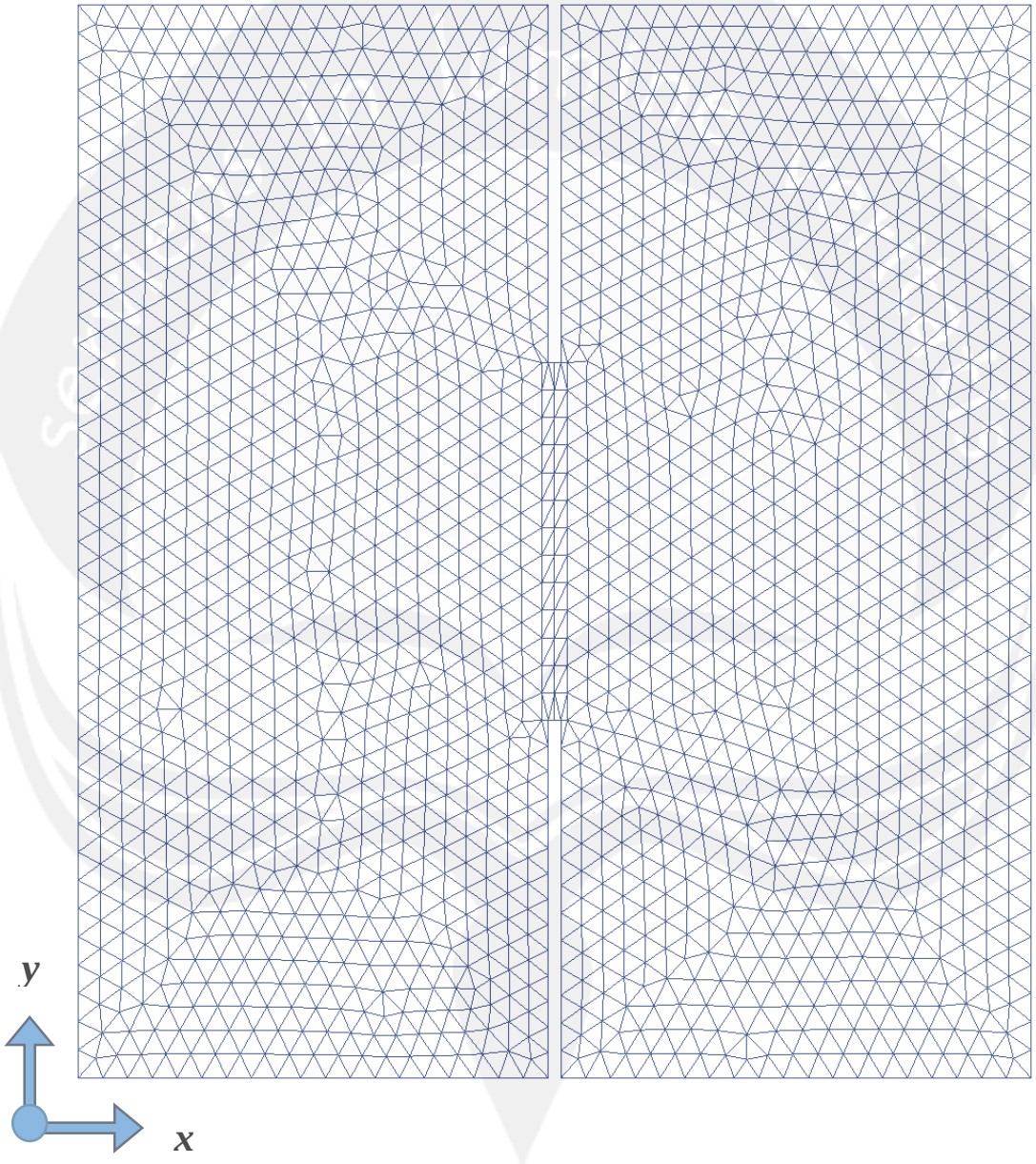
Keterangan (Jumlah)	Jenis Bencana					Total
	Banjir	Banjir dan Tanah Longsor	Gelom-bang Pasang / Abrasi	Gempa Bumi dan Tsunami	Tsu-nami	
Korban Meninggal	18.984	2.337	160	167.779	3.519	**192.779**
Korban Hilang	2.560	5.378	49	6.333	2.957	**17.277**
Korban Terluka	195.102	40.830	228	3.988	273	**240.421**
Pengungsi	5.350.289	505.852	29.410	462.272	238	**6.348.061**
Rumah Rusak Berat	83.882	13.169	3.461	325.157	18.017	**443.686**
Rumah Rusak Sedang	7.116	1.415	420	0	0	**8.951**
Rumah Rusak Ringan	168.220	26.654	3.866	97.403	17	**296.160**
Kerusakan Fasilitas Peribadatan	2.566	273	19	29	1	**2.888**
Kerusakan Fasilitas Pendidikan	6.311	1.062	26	1.262	114	**8.775**
Kerusakan Fasilitas Kesehatan	2.074	267	6	254	2	**2.603**
Kerusakan Jalan (kilometer)	68.607,15	1.178,52	410,99	34.904,2	0.5	**105.101**
Kerusakan Lahan (Ha)	1.499.441,63	292.183,38	317	58.087	216	**1.850.245**

Sumber : BNPB, 2016. Statistik - Data Bencana: Data dan Informasi Bencana Indonesia, Badan Nasional Penanggulangan Bencana (http://dibi.bnpb.go.id/data-bencana/statistik).

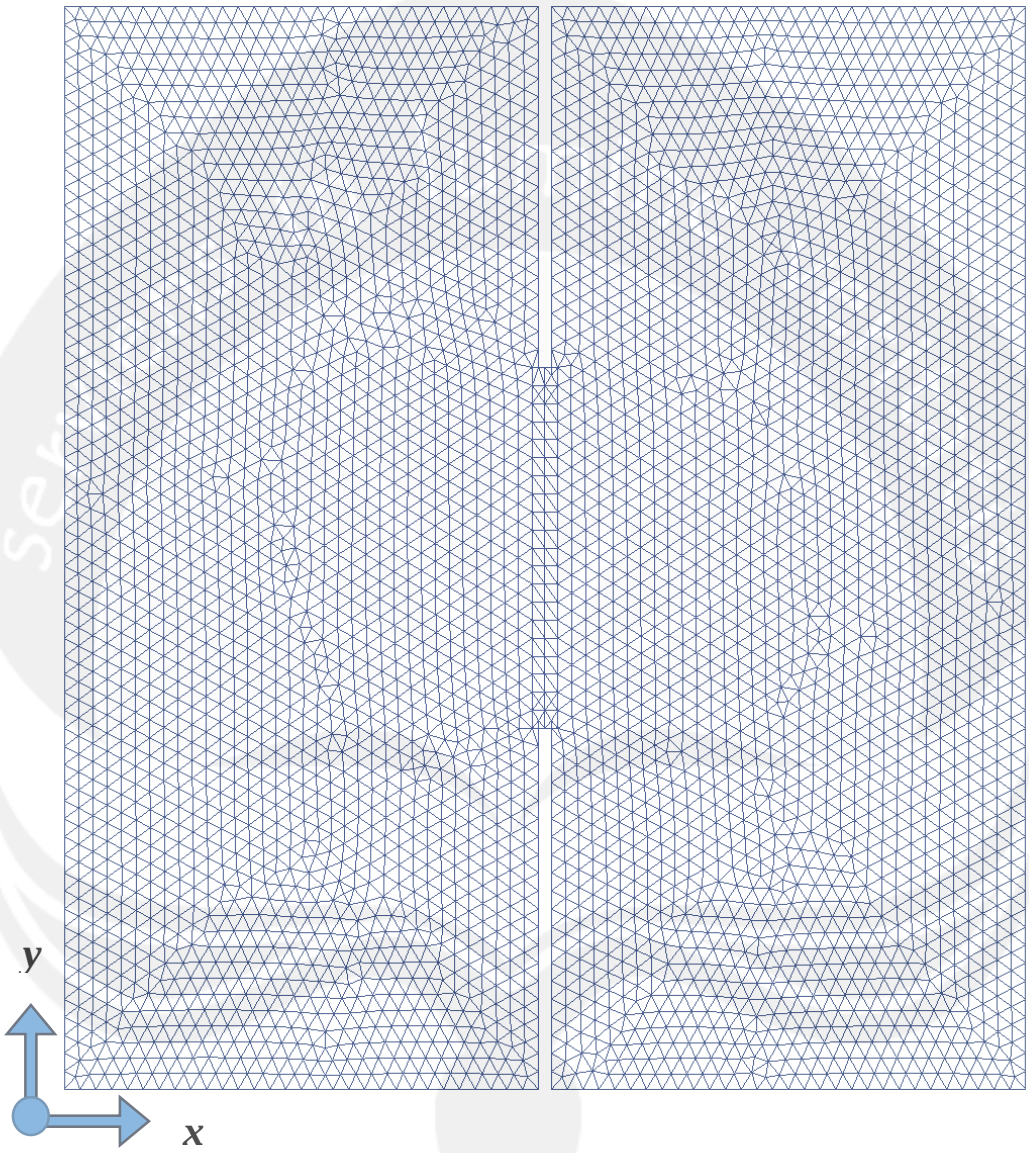
***Keterangan : Data pada kolom 'Total' adalah data yang telah diolah sendiri oleh penulis.**

LAMPIRAN B

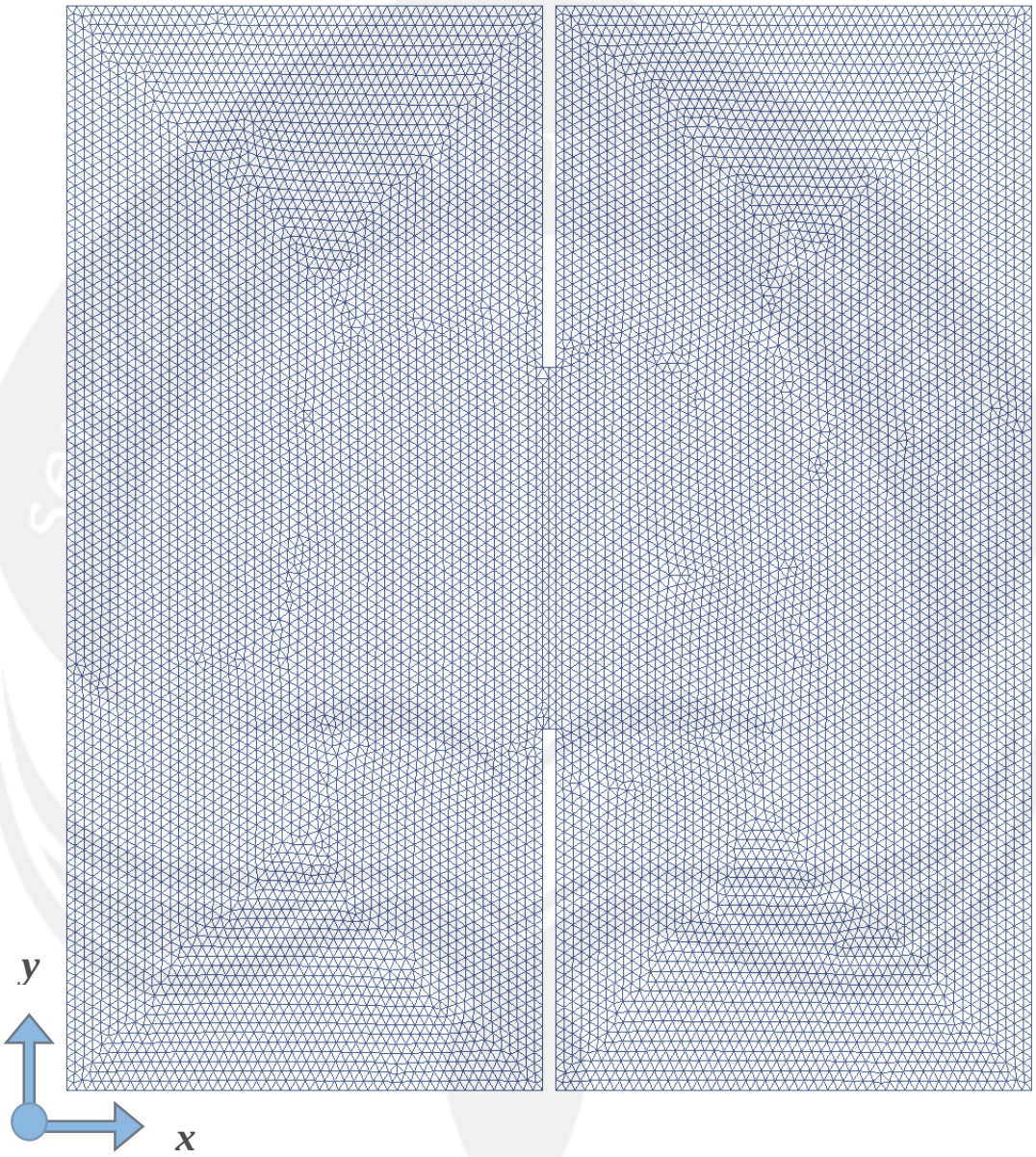
Gambar Bentuk dan Struktur Mesh



Gambar B.1. Bentuk dan struktur *mesh* DamBreak1.



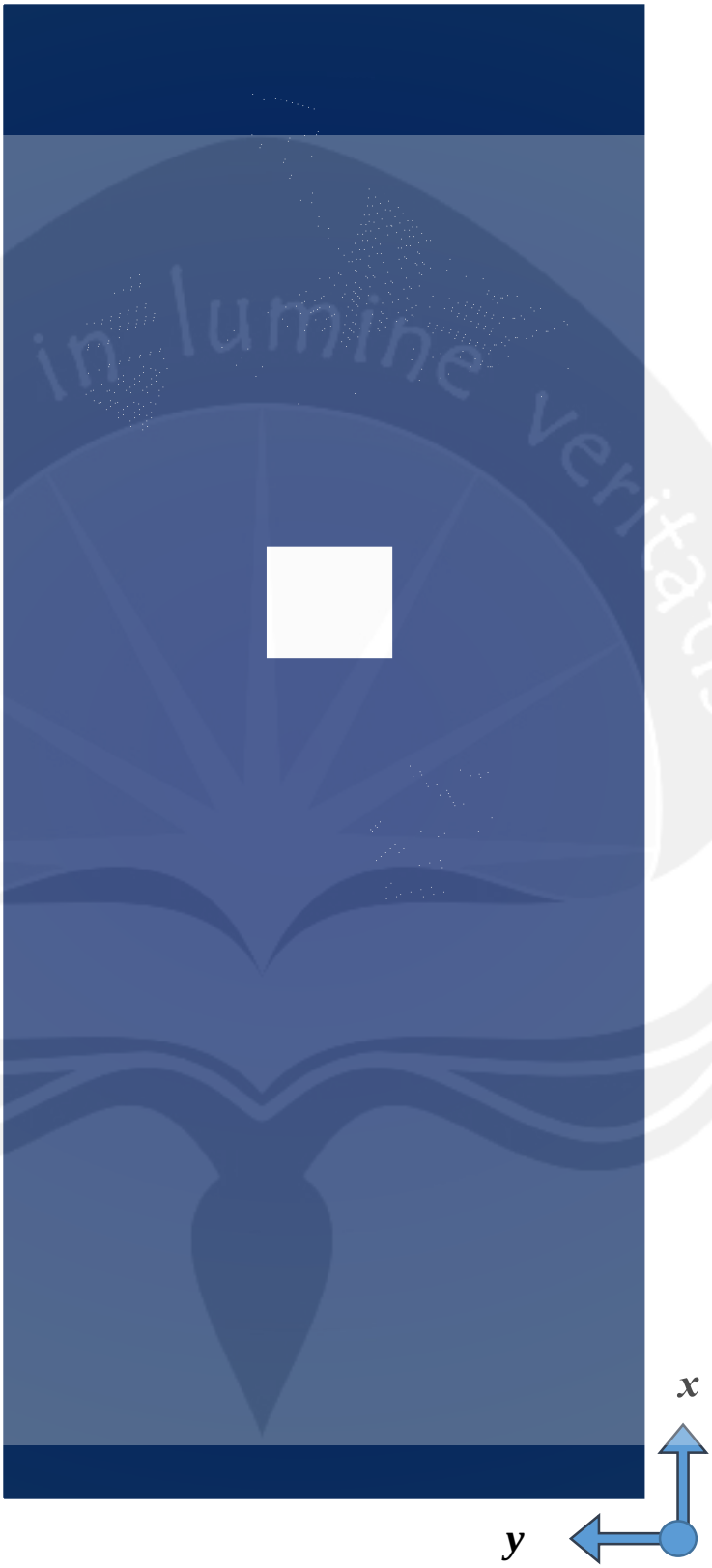
Gambar B.2. Bentuk dan struktur *mesh* DamBreak2.



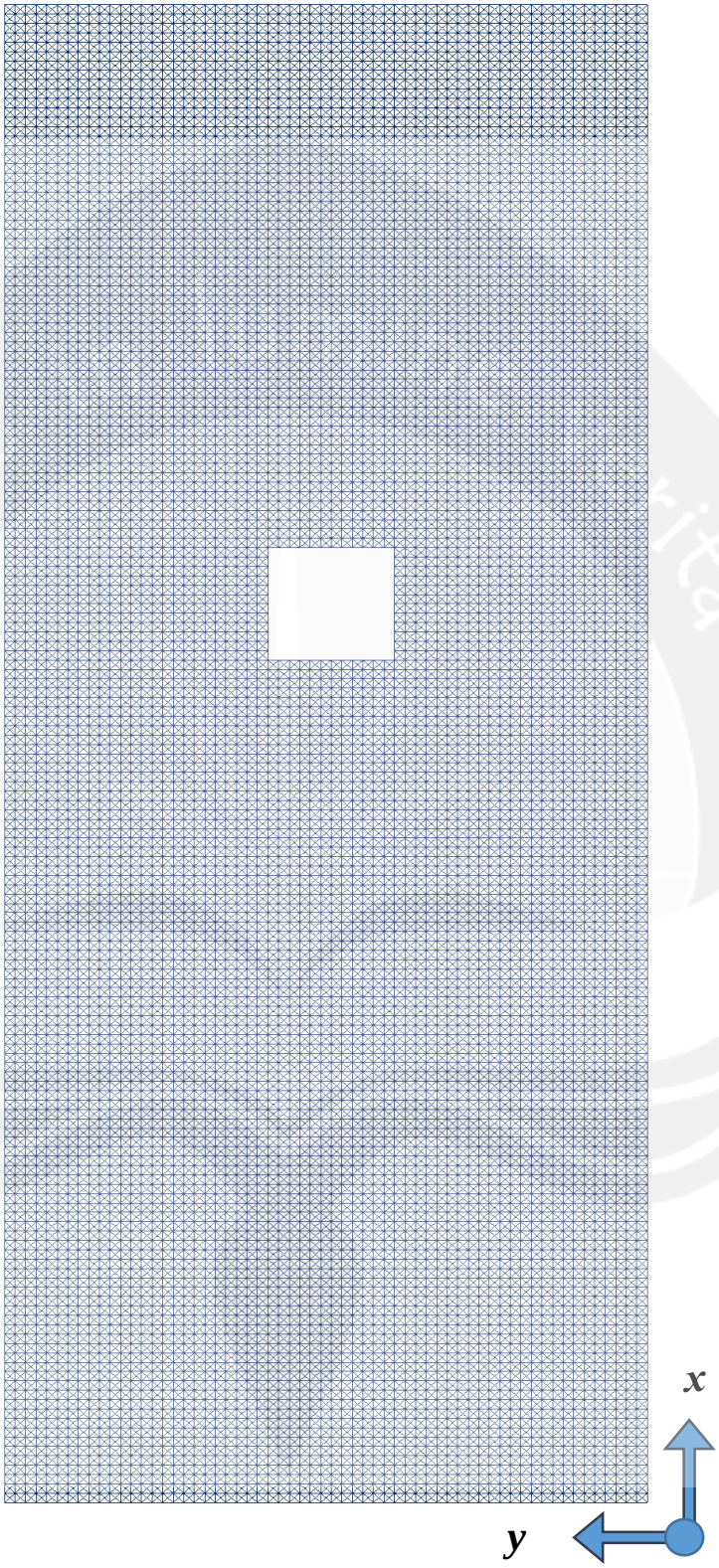
Gambar B.3. Bentuk dan struktur *mesh* DamBreak3.

Gambar B.4. Bentuk dan struktur *mesh* DamBreak4.

Gambar B.5. Bentuk dan struktur *mesh* DamBreak5.



Gambar B.6. Bentuk dan struktur *mesh* DamBreak6.



Gambar B.7. Bentuk dan struktur *mesh* DamBreak_Validasi.

LAMPIRAN C

Data Lengkap Perbandingan Waktu Komputasi GPU dan CPU

Tabel C.1. Perbandingan Waktu Komputasi untuk 200 Iterasi (dalam Sekon)

Jumlah Simpul	Jumlah elemen	Mesh	CPU	GPU2	GPU4	GPU8	GPU16	GPU32	GPU64	GPU128	GPU256	GPU512	GPU1024	Tercepat	SpeedUp Tercepat
1880	3548	DamBreak1	0.172	0.474	0.233	0.126	0.07	0.044	0.044	0.044	0.044	0.043	0.046	512	4.00
4348	8370	DamBreak2	0.403	1.038	0.551	0.296	0.162	0.108	0.081	0.078	0.08	0.081	0.084	128	5.20
11568	22600	DamBreak3	1.082	2.764	1.439	0.762	0.405	0.241	0.157	0.156	0.159	0.166	0.171	128	6.90
70176	139264	DamBreak4	7.138	17.401	9.129	4.866	2.623	1.497	0.872	0.871	0.898	0.937	0.955	128	8.20
279616	557056	DamBreak5	26.929	80.643	42.667	22.432	11.82	6.448	3.596	3.555	3.621	3.75	3.829	128	7.60
1116288	2228224	DamBreak6	101.984	345.126	174.067	89.443	46.132	24.979	13.688	13.55	13.627	14.033	14.268	128	7.50

Tabel C.2. Perbandingan Waktu Komputasi untuk 500 Iteration (dalam Sekon)

Jumlah Simpul	Jumlah elemen	Mesh	CPU	GPU2	GPU4	GPU8	GPU16	GPU32	GPU64	GPU128	GPU256	GPU512	GPU1024	Tercepat	SpeedUp Tercepat
1880	3548	DamBreak1	0.436	1.1	0.564	0.303	0.168	0.107	0.107	0.107	0.107	0.107	0.113	32, 64, 128, 256, 512	4.10
4348	8370	DamBreak2	0.997	2.541	1.319	0.713	0.384	0.253	0.188	0.186	0.188	0.195	0.204	128	5.40
11568	22600	DamBreak3	2.69	6.848	3.531	1.865	0.992	0.589	0.384	0.385	0.393	0.407	0.42	64	7.00
70176	139264	DamBreak4	17.092	48.755	26.147	13.968	7.412	4.099	2.311	2.294	2.336	2.415	2.456	128	7.40
279616	557056	DamBreak5	64.598	215.794	111.109	57.149	29.694	16.186	8.818	8.738	8.822	9.09	9.255	128	7.40
1116288	2228224	DamBreak6	255.449	885.073	443.937	227.696	116.407	63.326	34.442	34.092	34.219	35.148	35.732	128	7.50

LAMPIRAN D

Struktur dan Format File Mesh

File *mesh* yang digunakan adalah file dengan format dan aturan yang penulis definisikan sendiri untuk kepentingan penelitian. File ini menyimpan data-data *mesh* berupa simpul-simpul, elemen-elemen, dan dinding-dinding. File ini akan dibaca oleh progam pada saat *run-time* sebagai file bertipe teks. Berikut struktur dan format file *mesh* tersebut.

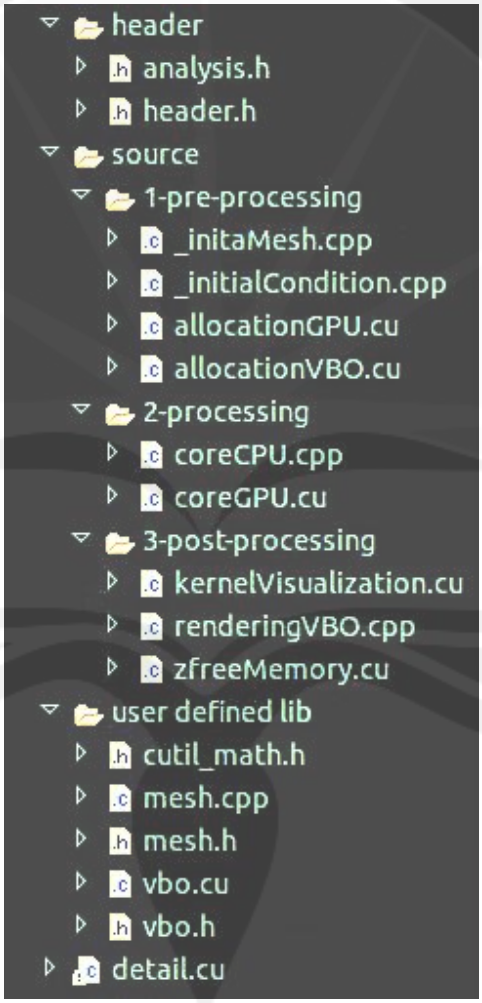
1. Enam baris pertama berupa karakter spasi.
2. Baris berikutnya berupa bilangan bulat *N*, *M* dan *O*. Masing-masing menyatakan jumlah simpul (*nodes*), jumlah elemen (sel) segitiga pada *mesh*, dan jumlah dinding (*wall*) yang ingin dibangun.
3. Dua baris berikutnya berupa karakter spasi.
4. Pada *N* baris berikutnya, tiap-tiap baris ke – *i* berisi bilangan *float* X_i dan Y_i, masing-masing menyatakan posisi (x, y) suatu simpul V_i dalam sistem koordinat Cartesian, dengan $i = 0, 1, \dots, N-1$.
5. Dua baris berikutnya berupa karakter spasi.
6. Pada M baris berikutnya, tiap-tiap baris ke – j berisi bilangan bulat P_j, Q_j, dan R_j, masing-masing menyatakan indeks (i) dari simpul-simpul (V_i) yang membentuk sebuah elemen segitiga T_j, dengan $j = 0, 1, \dots, M-1$, dan $0 \leq P_j, Q_j, R_j < N$.
7. Dua baris berikutnya berupa karakter spasi.
8. Masing-masing *O* baris berikutnya, terdiri dari bilangan *float* A_x, A_y, B_x, B_y, dan C_z. Bilangan A_x dan A_y merupakan koordinat *x* dan *y* titik *A*, serta B_x dan B_y merupakan koordinat *x* dan *y* titik *B*. Titik *A* dan B merupakan titik pada ujung-ujung garis *G*. Garis *G* merepresentasikan panjang sebuah dinding, dan C_z merepresentasikan ketinggian dinding tersebut. Sementara itu, ketebalan dinding diabaikan.

LAMPIRAN E

Kode Program

E.1. Struktur Kode Program

Kode program memiliki struktur seperti Gambar E.1. di bawah ini. Kode secara umum dikelompokkan menjadi tiga bagian yaitu *pre-processing*, *processing* dan *post-processing*.



Gambar E.1 Struktur kode program.

Selain itu, terdapat data-data lain dalam struktur kode program, yaitu dua buah file GLSL, *fragment shader* dan *vertex shader*. Contoh file *vertex shader* dan *fragment shader* yang digunakan dalam program ini yaitu **"test.vert"** dan **"test.frag".**

E.2. Alur Utama Program

Alur utama program terdapat dalam prosedur **mainFlow** dan **display**. Kedua prosedur tersebut bersama dengan **void main()** terdapat dalam file **Detail.cu.**

```
void mainFlow()
{
   InitMesh(&dataHost.Mesh, "DamBreak_Validasi");
   InitDataComputation(&dataHost,globalParam.dt);
   GPUSimulationAllocationMemory(&dataDev, dataHost);
   threadAllocation(globalParam.threadPerBlock,
            &GPUThread,dataDev);
   initVBO(&VBO,dataHost.Mesh);
   glutMainLoop();
}
```

```
void display()
{
   if(STATUS==RUNNING)
   {
      Analis.frame++;
      if(globalParam.device==DEVICE_GPU)
      {  gpuTimingStartRec(&Analis.gCoreTime);
         computeGPU(globalParam.iteration,GPUThread,&dataDev);
         gpuTimingStopRec(&Analis.gCoreTime,
                          Analis.gCoreTime.elapsed_time);
         Analis.simulationTime+=dataDev.dt*globalParam.iteration;

      }else
      {
         cpuTimingStartRec(&Analis.cCoreTime);
         computeCPU(globalParam.iteration, &dataHost);
         cpuTimingStopRec(&Analis.cCoreTime,
                          Analis.cCoreTime.elapsed_time);
         Analis.simulationTime+=dataHost.dt*globalParam.iteration;
      }
    }

   if(!(globalParam.style==STYLE_NOVISUALIZATION))
   {
      computeVisualization(VBO,globalParam,
                          GPUThread,&dataDev,dataHost);
//kontrol layar openGL
         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
         glMatrixMode(GL_MODELVIEW);
```

```
        glLoadIdentity();
        glTranslatef(0.0, 0.0, translate_z);//-3.25);
        glRotatef(rotate_x+5, 1.0, 0.0, 0.0);
        glRotatef(rotate_y -5, 0.0, 1.0, 0.0);
        glTranslatef(-0.5 + translate_x, -0.5, 0.75+translate_y);

       renderingVBO(shaderProg, VBO,dataDev,globalParam);
}

   if(STATUS==RUNNING)
    if(timesUp(globalParam.endtime,Analis.simulationTime) ||
       frameOver(globalParam.targetFrame,Analis.frame))
      {
       cpuTimingStopRec(&Analis.cVisualTime,
       Analis.cVisualTime.elapsed_time);
       STATUS=PAUSE;
      }

    glutSwapBuffers();
}
```

E.3. File header.h

```
#ifndef HEADER_H_
#define HEADER_H_

#include <X11/Xlib.h>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <sys/time.h>
#include <vector_types.h>
#include <cuda_runtime.h>
#include <cuda_gl_interop.h>
#include <helper_functions.h>
#include <helper_cuda.h>
#include <helper_cuda_gl.h>
#include "analysis.h"
#include "../user defined lib/mesh.h"
#include "../user defined lib/cutil_math.h"

#define g 9.8
#define MAX(a,b) ((a > b) ? a : b)
#define PAUSE 0
#define RUNNING 1

#define DEVICE_GPU 1
#define DEVICE_CPU 0
#define STYLE_NOVISUALIZATION 0
#define STYLE_SURFACE 1
#define STYLE_WIRE 2
#define STYLE_COLORHSV 3
#define TRUE 1
#define FALSE 0

typedef struct{
```

```
   double *hnew, *unew, *vnew, *hunew, *hvnew;
   double *h, *u, *v, *hu, *hv,*z;
   double dt;
   mesh Mesh;

}dataWaveCompute;

typedef struct{
   GLuint
    posVertexBuffer,
    indexBuffer,
    VertexNormalBuffer,

    signBuffer;

   struct cudaGraphicsResource
        *cuda_VertexWall_resource,
        *cuda_VertexPos_resource,
        *cuda_VertexNormal_resource,
         *cuda_Sign_resource;
}vbo;


typedef struct{
   dim3 block;     dim3 grid1; dim3 grid2;
}gpuThread;

typedef struct
{  float4 *dptr;   float3 *nptr;     float *sptr;
}vboPointer;

// Initial Mesh
int InitMesh(mesh * Mesh, char *title);

//initial condition
int InitDataComputation(dataWaveCompute *data,double dt);

//Allocation Memory
double GPUSimulationAllocationMemory
       (dataWaveCompute *dataDev, dataWaveCompute dataHost);
int threadAllocation
       (int blockAllocation, gpuThread *GPUThread,
        dataWaveCompute dataDev);

//init vbo
void initVBO(vbo *VBO, mesh Mesh);

//Processing
double computeGPU (int iteration ,
       gpuThread GPUThread,   dataWaveCompute *dataDev);
double computeCPU(int iteration,
       dataWaveCompute  *dataHost);

//Post Processing
void computeVisualization(vbo VBO,  parameter Param,
```

```
      gpuThread  GPUThread,  dataWaveCompute *dataDev,
      dataWaveCompute dataHost);
void renderingVBO(GLuint shaderProg, vbo VBO, dataWaveCompute
dataDev, parameter Param);


//Free Memory
double GPUSimulationFreeMemory(dataWaveCompute *dataDev);
void cleanup();

//Event
// rendering callbacks
void display();
void keyboard(unsigned char key, int x, int y);
void mouse(int button, int state, int x, int y);
void motion(int x, int y);
void timerEvent(int value);

// fungsi OpenGL
bool initGL(int *argc, char **argv);
// buatan penulis untuk batas waktu/frame simulasi
int timesUp( double endTime,double elapsedSimulation);
int frameOver( int targetFrame,int frame);

// bawaan template
bool runTest(int argc, char **argv, char *ref_file);

#endif /* HEADER_H_ */
```

E.4. File analysis.h

```
#include <cuda_runtime.h>

typedef struct{
  short int device;
  short int style;
  short int threadPerBlock;
  double dt;
  char *sourceFile;
  char *outFile;
  int iteration;
  int targetFrame;
  double endtime;
}parameter;

typedef struct{
  cudaEvent_t start, stop;
  float elapsed_time;
}gpuTiming;


typedef struct{
  double start, stop;
  double elapsed_time;
```

```
}cpuTiming;

typedef struct{
   short int thread;
   int iteration;
   gpuTiming gCoreTime;
   cpuTiming cCoreTime;
   cpuTiming cVisualTime;
   unsigned int frame;
   double simulationTime;
}dataAnalysis;

double cpuSecond();
void gpuTimingStartRec(gpuTiming *GPUTiming);
void gpuTimingStopRec(gpuTiming *GPUTiming, float prevElapsed);
void cpuTimingStartRec(cpuTiming *CPUTiming);
void cpuTimingStopRec(cpuTiming *CPUTiming,float prevElapsed);
void initParam(parameter *Param);
void initAnalis(dataAnalysis *Analis);
void showAnalis(parameter P, dataAnalysis A);
```

E.5. File _initaMesh.cpp

```
#include "../../header/header.h"

int InitMesh(mesh * Mesh, char *title)
{
   readMeshFile(Mesh,title);
   setMeshRelation(Mesh);
   setMeshInformation(Mesh);
   return 1;

}
```

E.6. File _initialCondition.cpp

```
#include "../../header/header.h"

int InitDataComputation(dataWaveCompute *data, double dt)
{
   data->h=dvector(data->Mesh.NCells);
   data->u=dvector(data->Mesh.NCells);
   data->v=dvector(data->Mesh.NCells);
   data->hu=dvector(data->Mesh.NCells);
   data->hv=dvector(data->Mesh.NCells);
   data->hnew=dvector(data->Mesh.NCells);
   data->unew=dvector(data->Mesh.NCells);
   data->vnew=dvector(data->Mesh.NCells);
   data->hunew=dvector(data->Mesh.NCells);
   data->hvnew=dvector(data->Mesh.NCells);
   data->dt=dt;

   for(int k=0;k<data->Mesh.NCells;++k){
```

```
    if(data->Mesh.CX[k]<=0.4)
    {
     data->h[k]=0.3;
    }
    else if (data->Mesh.CX[k]<=0.4+0.5)
    {
     data->h[k]=0.01;
    }
    else  if(data->Mesh.CX[k]<0.4+0.5+0.12)
    {
     if(data->Mesh.CY[k]>=0.25 && data->Mesh.CY[k]<=0.25+0.12)
     {
        data->h[k]=0.75;
     }
     else
        data->h[k]=0.01;
    }else
    data->h[k]=0.01;
    data->u[k]=0.0;
    data->v[k]=0.0;
    data->hu[k]=0.0;
    data->hv[k]=0.0;
   }
 return 1;
}
```

E.7. File allocationGPU.cu

```
#include "../../header/header.h"

double GPUSimulationAllocationMemory(dataWaveCompute *dataDev,
    dataWaveCompute dataHost){

    cudaSetDevice(0);
    dataDev->dt=dataHost.dt;
    dataDev->Mesh=dataHost.Mesh;

    cudaMalloc((double**)&dataDev->h,
                dataDev->Mesh.NCells * sizeof(double));
    cudaMalloc((double**)&dataDev->u,
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->v,
                 dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->hu,
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->hv,
                dataDev->Mesh.NCells* sizeof(double));

    cudaMalloc((double**)&dataDev->hnew,
                dataDev->Mesh.NCells * sizeof(double));
    cudaMalloc((double**)&dataDev->unew,
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->vnew,
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->hunew,
```

```
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->hvnew,
                dataDev->Mesh.NCells* sizeof(double));

    cudaMalloc((double**)&dataDev->Mesh.L,
                dataDev->Mesh.NCells *3* sizeof(double));
    cudaMalloc((int**)&dataDev->Mesh.EtoE,
                dataDev->Mesh.NCells *3* sizeof(int));
    cudaMalloc((int**)&dataDev->Mesh.EtoV,
                dataDev->Mesh.NCells *3* sizeof(int));
    cudaMalloc((int**)&dataDev->Mesh.VtoE, dataDev->Mesh.NNodes *
                dataDev->Mesh.maxVtoEconn* sizeof(int));
    cudaMalloc((int**)&dataDev->Mesh.VtoS, dataDev->Mesh.NNodes *
                dataDev->Mesh.maxVtoEconn* sizeof(int));


    cudaMalloc((double**)&dataDev->Mesh.nx,
                dataDev->Mesh.NCells *3* sizeof(double));
    cudaMalloc((double**)&dataDev->Mesh.ny,
                dataDev->Mesh.NCells *3* sizeof(double));
    cudaMalloc((double**)&dataDev->Mesh.AREA,
                dataDev->Mesh.NCells* sizeof(double));
    cudaMalloc((double**)&dataDev->Mesh.VertX,
                dataDev->Mesh.NNodes* sizeof(double));
    cudaMalloc((double**)&dataDev->Mesh.VertY,
                dataDev->Mesh.NNodes* sizeof(double));
    cudaMalloc((double**)&dataDev->Mesh.NormalFaces,
                dataDev->Mesh.NCells* sizeof(float3));

    cudaMemcpy(dataDev->h, dataHost.h, dataDev->Mesh.NCells *
                sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->u, dataHost.u, dataDev->Mesh.NCells *
                sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->v, dataHost.v, dataDev->Mesh.NCells *
                sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->hu, dataHost.hu, dataDev->Mesh.NCells *
                sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->hv, dataHost.hv, dataDev->Mesh.NCells *
                sizeof(double), cudaMemcpyHostToDevice);

    cudaMemcpy(dataDev->Mesh.L, dataHost.Mesh.L,
                dataDev->Mesh.NCells * 3*sizeof(double),
                      cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.EtoE, dataHost.Mesh.EtoE,
                dataDev->Mesh.NCells * 3*sizeof(int),
                      cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.EtoV, dataHost.Mesh.EtoV,
                dataDev->Mesh.NCells * 3*sizeof(int),
                      cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.VtoE, dataHost.Mesh.VtoE,
                dataDev->Mesh.NNodes *
                      dataDev->Mesh.maxVtoEconn*sizeof(int),
                            cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.VtoS, dataHost.Mesh.VtoS,
                dataDev->Mesh.NNodes *
                      dataDev->Mesh.maxVtoEconn*sizeof(int),
```

```
                                cudaMemcpyHostToDevice);

    cudaMemcpy(dataDev->Mesh.nx, dataHost.Mesh.nx,
                    dataDev->Mesh.NCells * 3*sizeof(double),
                          cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.ny, dataHost.Mesh.ny,
                    dataDev->Mesh.NCells * 3*sizeof(double),
                          cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.AREA, dataHost.Mesh.AREA,
                    dataDev->Mesh.NCells * sizeof(double),
                          cudaMemcpyHostToDevice);


    cudaMemcpy(dataDev->Mesh.VertX, dataHost.Mesh.VertX,
                    dataDev->Mesh.NNodes * sizeof(double),
                          cudaMemcpyHostToDevice);
    cudaMemcpy(dataDev->Mesh.VertY, dataHost.Mesh.VertY,
                    dataDev->Mesh.NNodes * sizeof(double),
                          cudaMemcpyHostToDevice);
    return 1.0;

}

int threadAllocation(int blockAllocation, gpuThread *GPUThread,
   dataWaveCompute dataDev)
{
   dim3 block(blockAllocation);
   dim3 grid1 ((dataDev.Mesh.NCells+ block.x-1)/block.x);
   dim3 grid2((dataDev.Mesh.NCells+ block.x-1)/block.x);

   GPUThread->grid1=grid1;
   GPUThread->grid2=grid2;
   GPUThread->block=block;

   return 1;
}
```

E.8. File allocationVBO.cu

```
#include "../../header/header.h"
#include "../../user defined lib/vbo.h"

void createMeshIndexBuffer(GLuint *id, mesh Mesh)
    //int *EtoV, int Nelems, int NNodes)
{
    int i, size = (Mesh.NCells+Mesh.Wall.count*6)*3*
                  sizeof(GLuint);

    int endPosVertex=Mesh.NNodes;

    // create index buffer
    glGenBuffersARB(1, id);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, *id);
    glBufferDataARB(GL_ELEMENT_ARRAY_BUFFER, size, 0,
       GL_STATIC_DRAW);
```

```
   GLuint *indices = (GLuint *) glMapBuffer(
                                    GL_ELEMENT_ARRAY_BUFFER,
                                    GL_WRITE_ONLY);
    if (!indices)
    {
        return;
    }

    for (i=0; i<Mesh.NCells*3; i++)
    {
     *indices++=Mesh.EtoV[i];
    }
    for(i=0; i<Mesh.Wall.count*6;i++)
    {
     *indices++=endPosVertex++;
    }
    glUnmapBuffer(GL_ELEMENT_ARRAY_BUFFER);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

}
```

```
void createMeshPositionVBO(GLuint *id, struct cudaGraphicsResource
**vertex_pos, mesh Mesh)
{
    createVBO2(id, (Mesh.NNodes + Mesh.Wall.count*6)*4*
                                sizeof(float));
    glBindBuffer(GL_ARRAY_BUFFER, *id);
    float *pos = (float *) glMapBuffer( GL_ARRAY_BUFFER,
                                        GL_WRITE_ONLY);

    if (!pos)
    {
        return;
    }

    for (int idx=0; idx<Mesh.NNodes; idx++)
    {
        *pos++ = Mesh.VertX[idx];
        *pos++ = 0.0f;
        *pos++ = Mesh.VertY[idx];
        *pos++ = 1.0f;
    }

    for (int idx=0; idx<Mesh.Wall.count; idx++)
    {
       float3 v0 =make_float3 (Mesh.Wall.Point1[idx].x,
       Mesh.Wall.Point1[idx].y,0);
       float3 edge1=Mesh.Wall.Point2[idx]-Mesh.Wall.Point1[idx];
       float3 edge2=v0-Mesh.Wall.Point1[idx];
       float3 normal=cross(edge1,edge2);

            *pos++ =  Mesh.Wall.Point1[idx].x;
            *pos++ = 0.0f;
            *pos++ = Mesh.Wall.Point1[idx].y;
```

```
            *pos++=1.0;

            *pos++ =  Mesh.Wall.Point1[idx].x;
            *pos++ = Mesh.Wall.Point1[idx].z;
            *pos++ = Mesh.Wall.Point1[idx].y;
            *pos++=1.0;

            *pos++ =  Mesh.Wall.Point2[idx].x;
            *pos++ = Mesh.Wall.Point2[idx].z;
            *pos++ = Mesh.Wall.Point2[idx].y;
            *pos++=1.0;

            *pos++ =  Mesh.Wall.Point1[idx].x;
            *pos++ = 0.0f;
            *pos++ = Mesh.Wall.Point1[idx].y;
            *pos++=1.0;

            *pos++ =  Mesh.Wall.Point2[idx].x;
            *pos++ = 0.0f;
            *pos++ = Mesh.Wall.Point2[idx].y;

            *pos++=1.0;
            *pos++ =  Mesh.Wall.Point2[idx].x;
            *pos++ = Mesh.Wall.Point2[idx].z;
            *pos++ = Mesh.Wall.Point2[idx].y;
            *pos++=1.0;
    }

    glUnmapBuffer(GL_ARRAY_BUFFER);
    glBindBuffer(GL_ARRAY_BUFFER, 0);


    checkCudaErrors(cudaGraphicsGLRegisterBuffer(vertex_pos, *id,
                        cudaGraphicsMapFlagsNone));
}
```

```
void initVBO(vbo *VBO, mesh Mesh)
{
   createVBO2(&VBO->VertexNormalBuffer, (Mesh.NNodes+
                        Mesh.Wall.count*6) *sizeof(float3));

   glBindBuffer(GL_ARRAY_BUFFER, VBO->VertexNormalBuffer);
   float3 *pos = (float3 *)
                 glMapBuffer(GL_ARRAY_BUFFER, GL_WRITE_ONLY);

  if (!pos)
  {
    return;
  }else
  {
     pos=pos+Mesh.NNodes;
    for(int ii=0;ii<Mesh.Wall.count;ii++)
    {
       int s=-1;
```

```
        if(ii%3<3)s*=-1;
        float3 v0=make_float3 (Mesh.Wall.Point1[ii].x,
                               Mesh.Wall.Point1[ii].y,0);
         float3 edge1=Mesh.Wall.Point2[ii]-Mesh.Wall.Point1[ii];
        float3 edge2=v0-Mesh.Wall.Point1[ii];

        float3 normal=make_float3(0,0,0);//cross(edge2,edge1);

        for(int i=0;i<6;i++)
             *pos++=normal;
      }
  }
  glUnmapBuffer(GL_ARRAY_BUFFER);
  glBindBuffer(GL_ARRAY_BUFFER, 0);


  checkCudaErrors(cudaGraphicsGLRegisterBuffer(
                  &VBO->cuda_VertexNormal_resource,
                       VBO->VertexNormalBuffer,
                             cudaGraphicsMapFlagsWriteDiscard));

  createVBO2(&VBO->signBuffer, Mesh.NNodes*sizeof(float));

  checkCudaErrors(cudaGraphicsGLRegisterBuffer (
                  &VBO-> cuda_Sign_resource,
                       VBO->signBuffer,
                             cudaGraphicsMapFlagsWriteDiscard));

  createMeshPositionVBO(&VBO->posVertexBuffer,
                             &VBO-> cuda_VertexPos_resource,
                                   Mesh);

  createMeshIndexBuffer(&VBO->indexBuffer, Mesh);

}
```

E.9. File coreCPU.cpp

```
#include "../../header/header.h"


double max(double a, double b)
{
  if(a>b)return a;
  else return b;

}
double min(double a, double b)
{
  if(a<b)return a;
  else return b;
}
double calculate_a(double *a_plus, double *a_min, double uj,
  double uk, double hj, double hk)
{
```

```
   *a_plus=max(uj+sqrt(g*hj),uk+sqrt(g*hk));
   *a_plus=max(*a_plus,0.0);
   *a_min=min(uj-sqrt(g*hj),uk-sqrt(g*hk));
   *a_min=min(*a_min,0.0);
   return 1;
}

double calculate_H(double HUj, double HUk, double a_plus, double
a_min, double Uj,double Uk)
{
   double da=a_plus-a_min;
   if(da<10e-8) return 0.5*(HUj-HUk);
   return ((a_plus*HUj - a_min*HUk)/da + a_plus*a_min/da*(Uk-Uj));

}

double calculate_Unew(double Uold, double EFluxCrossL, double dt,
double A)
{
   return (Uold-dt*EFluxCrossL/A);
}
```

```
double kernelCPU(int iteration, float *Sign, int *EtoV,
    double *h1, double *u1, double *v1, double *hu1, double *hv1,
    double *h, double *u, double *v, double *hu, double *hv,
    double *L, int *EtoE, double *normx, double *normy, double
*AREA, double dt, int Nelems)

{

   double hj, hk, uj, uk, vj,vk, huj, huk, hvj, hvk;
   double a_plus, a_min;
   double HUj, HUk, nx, ny, l, A;
   int look=0;

   double start_time;
   start_time = cpuSecond();

   for(int it=0;it<iteration; it++)
   {
    for(int idx=0; idx<Nelems;idx++)
    {
     double Flux_h=0.0, Flux_hu=0.0, Flux_hv=0.0;
     //fill common elements
     hj=h[idx];
     uj=u[idx];
     vj=v[idx];
     huj=hu[idx];
     hvj=hv[idx];
     A=AREA[idx];
     uj=huj/hj;
     vj=hvj/hj;
```

```
 for(int n=0; n<3;n++)
 {
   int k = idx*3+ n;
    int nb= EtoE[k];

    nx=normx[k];
    ny=normy[k];
    l=L[k];

   hk=h[nb];
   uk=u[nb];
   vk=v[nb];
   huk=hu[nb];
   hvk=hv[nb];
   uk=huk/hk;
   vk=hvk/hk;

    if(nb==idx)
    {
        look=1;
        hk=hj;
        uk=-uj;
        vk=-vj;
        huk=-huj;
        hvk=-hvj;
    }
    /
//Calculate wave speed
   calculate_a(&a_plus, &a_min, nx*uj + ny*vj, nx*uk + ny*vk,
                    hj, hk);

    //Calculate Height Flux
    HUj= huj*nx + hvj*ny;
    HUk= huk*nx + hvk*ny;
    Flux_h+=l*calculate_H(HUj, HUk, a_plus, a_min, hj, hk);

    //Calculate X momentum Flux
    HUj = (huj*uj + 0.5*g*hj*hj)*nx;
    HUj+= (huj*vj)*ny;
    HUk = (huk*uk + 0.5*g*hk*hk)*nx;
    HUk+= (huk*vk)*ny;
    Flux_hu+=l*calculate_H(HUj, HUk, a_plus,a_min, huj, huk);

    //Calculate Y momentum Flux
    HUj = (hvj*uj)*nx;
    HUj+= (hvj*vj + 0.5*g*hj*hj)*ny;
    HUk = (hvk*uk)*nx;
    HUk+= (hvk*vk + 0.5*g*hk*hk)*ny;
    Flux_hv+=l*calculate_H(HUj, HUk,a_plus,a_min, hvj,  hvk);
 }

 h1[idx]=calculate_Unew(hj,Flux_h, dt,A) ;
 hu1[idx]=calculate_Unew(huj, Flux_hu,dt,A);
 hv1[idx]=calculate_Unew(hvj, Flux_hv,dt,A);
}
```

```
    for(int idx=0; idx<Nelems;idx++)
    {
     h[idx]=h1[idx];
     hu[idx]=hu1[idx];
     hv[idx]=hv1[idx];
     u[idx]=hu[idx]*1.0/(h[idx]*1.0);
     v[idx]=hv[idx]*1.0/(h[idx]*1.0);
    }
   }
   return cpuSecond()-start_time;
}
```

```
double computeCPU(int iteration, dataWaveCompute  *dataHost)
{
   float *Sign;
   kernelCPU(iteration, Sign, dataHost->Mesh.EtoV,
      dataHost->hnew, dataHost->unew, dataHost->vnew,
      dataHost->hunew, dataHost->hvnew,
      dataHost->h, dataHost->u, dataHost->v,
      dataHost->hu, dataHost->hv, dataHost->Mesh.L,
      dataHost->Mesh.EtoE, dataHost->Mesh.nx, dataHost->Mesh.ny,
     dataHost->Mesh.AREA, dataHost->dt, dataHost->Mesh.NCells);
   return 0;
}
```

E.10. File coreGPU.cu

```
#include "../../header/header.h"

inline __host__ __device__ double calculate_a(double *a_plus,
double *a_min, double uj, double uk, double hj, double hk)
{
   *a_plus=max(uj+sqrt(g*hj),uk+sqrt(g*hk));
   *a_plus=max(*a_plus,0.0);
   *a_min=min(uj-sqrt(g*hj),uk-sqrt(g*hk));
   *a_min=min(*a_min,0.0);
   return 1;
}

inline __host__ __device__ double calculate_H(double HUj, double
HUk, double a_plus, double a_min, double Uj,double Uk)
{
   double da=a_plus-a_min;
   if(da<10e-8) return 0.5*(HUj-HUk);
   return ((a_plus*HUj - a_min*HUk)/da + a_plus*a_min/da*(Uk-Uj));

}

inline __host__ __device__ double calculate_Unew(double Uold,
double EFluxCrossL, double dt, double A)
{
   return (Uold-dt*EFluxCrossL/A);
}
```

```
__global__ void kernelcopy(
    double *h1, double *u1, double *v1, double *hu1, double *hv1,
    double *h, double *u, double *v, double *hu, double *hv, int
Nelems)
{
   unsigned int idx=threadIdx.x + blockIdx.x * blockDim.x;

   if(idx>=Nelems) return;

   h[idx]=h1[idx];
   hu[idx]=hu1[idx];
   hv[idx]=hv1[idx];

   u[idx]=hu[idx]*1.0/(h[idx]*1.0);
   v[idx]=hv[idx]*1.0/(h[idx]*1.0);
}
```

```
__global__ void kernelFluxCalculation( int *EtoV,
    double *h1, double *u1, double *v1, double *hu1, double *hv1,
    double *h, double *u, double *v, double *hu, double *hv,

    double *L, int *EtoE, double *normx, double *normy, double
*AREA, double dt, int Nelems)

{
   unsigned int idx=threadIdx.x + blockIdx.x*blockDim.x;
   if(idx>=Nelems) return;

   double hj, hk, uj, uk, vj,vk, huj, huk, hvj, hvk;
   double a_plus, a_min;
   double HUj, HUk, nx, ny, l, A;
   double Flux_h=0.0, Flux_hu=0.0, Flux_hv=0.0;


   //fill common elements
   hj=h[idx];
   uj=u[idx];
   vj=v[idx];
   huj=hu[idx];
   hvj=hv[idx];
   A=AREA[idx];
   uj=huj/hj;
   vj=hvj/hj;

   for(int n=0; n<3;n++)
   {
      int k = idx*3+ n;
      int nb= EtoE[k];
      nx=normx[k];

      ny=normy[k];
      l=L[k];

      hk=h[nb];
      uk=u[nb];
```

```
      vk=v[nb];
      huk=hu[nb];
      hvk=hv[nb];
      uk=huk/hk;
      vk=hvk/hk;

      if(nb==idx)
      {
        hk=hj;
        uk=-uj;
        vk=-vj;
        huk=-huj;
        hvk=-hvj;

      }
      //Calculate wave speed
      calculate_a(&a_plus, &a_min,nx*uj +ny*vj, nx*uk+ny*vk,hj,hk);

      //Calculate Height Flux
      HUj= huj*nx + hvj*ny;
      HUk= huk*nx + hvk*ny;
      Flux_h+=l*calculate_H(HUj, HUk, a_plus, a_min, hj, hk);

      //Calculate X momentum Flux
      HUj = (huj*uj + 0.5*g*hj*hj)*nx;
      HUj+= (huj*vj)*ny;
      HUk = (huk*uk + 0.5*g*hk*hk)*nx;
      HUk+= (huk*vk)*ny;
      Flux_hu+=l*calculate_H(HUj, HUk, a_plus, a_min, huj, huk);

      //Calculate Y momentum Flux
      HUj = (hvj*uj)*nx;
      HUj+= (hvj*vj + 0.5*g*hj*hj)*ny;
      HUk = (hvk*uk)*nx;
      HUk+= (hvk*vk + 0.5*g*hk*hk)*ny;
      Flux_hv+=l*calculate_H(HUj, HUk, a_plus, a_min, hvj,  hvk);
   }

   h1[idx]=calculate_Unew(hj,Flux_h, dt,A) ;
   hu1[idx]=calculate_Unew(huj, Flux_hu,dt,A);
   hv1[idx]=calculate_Unew(hvj, Flux_hv,dt,A);
}
```

```
double computeGPU(int iteration , gpuThread GPUThread,
dataWaveCompute *dataDev)
{

   for(int i=0;i<iteration;i++)
   {

    kernelFluxCalculation<<<GPUThread.grid1,GPUThread.block>>>
             (dataDev->Mesh.EtoV, dataDev->hnew, dataDev->unew,
              dataDev->vnew, dataDev->hunew, dataDev->hvnew,
              dataDev->h, dataDev->u, dataDev->v, dataDev->hu,
```

```
              dataDev->hv, dataDev->Mesh.L, dataDev->Mesh.EtoE,
              dataDev->Mesh.nx, dataDev->Mesh.ny,
              dataDev->Mesh.AREA, dataDev->dt,
              dataDev->Mesh.NCells);

    kernelcopy<<<GPUThread.grid1,GPUThread.block>>>
             (dataDev->hnew,dataDev->unew,
              dataDev->vnew, dataDev->hunew, dataDev->hvnew,
              dataDev->h, dataDev->u, dataDev->v, dataDev->hu,
              dataDev->hv,dataDev->Mesh.NCells);
  }
 return 0;
}
```

E.11. File kernelVisualization.cu

```
#include "../../header/header.h"

__global__ void kernelUpdateVertex (float4 *pos,int *VtoE,
     int *VtoS, double *VertX, double *VertY, double *p ,
    int maxconn, int Nnodes)
{
  unsigned int idx=threadIdx.x + blockIdx.x * blockDim.x;
  double w;

  int i, k;
  float sum=0;
  int ie;
  int iconn=0;
  if(idx>=Nnodes) return;

  k=idx*maxconn;
  for(i=0;  i < maxconn; i++)
  {

   ie = VtoE[k+i];
   if(ie==-1)break;
   sum=sum+p[ie];
   iconn++;
  }

  w=1.0*sum/iconn;
  pos[idx]=make_float4(pos[idx].x,w,pos[idx].z, .0f);
}
```

```
__global__ void kernelNormalFace(float4 *vertexPos, float3
*normalFace, int *EtoV, int Nelems)
{

  float3 edge1,edge2;
  int i1,i2,i3;

  unsigned int idx=threadIdx.x + blockIdx.x * blockDim.x;
```

```
   if(idx>=Nelems) return;

   i1 = EtoV[idx*3 + 0];
   i2 = EtoV[idx*3 + 1];
   i3 = EtoV[idx*3 + 2];

   edge1=operator-(make_float3(vertexPos[i2]),
         make_float3(vertexPos[i1]));
   edge2=operator-(make_float3(vertexPos[i3]),
         make_float3(vertexPos[i1]));

   normalFace[idx] = cross(edge2,edge1);
}
```

```
__global__ void kernelNormalVektor(float4 *vertexPos, float3
*normalVector, float3 *normalFace, int *VtoE, int maxconn, int
Nnodes)
{
   unsigned int idx=threadIdx.x + blockIdx.x * blockDim.x;
   float3 vsum=make_float3(0.0,0.0,0.0);
   int i, k=idx*maxconn;
   if(idx>=Nnodes) return;

    for(i=0;  i < maxconn; i++)
    {

     int ie = VtoE[k+i];
     if(ie==-1)break;
     operator+=(vsum, normalFace[ie]);

    }

    normalVector[idx]=operator/(vsum,i);
}
```

```
void computeVisualization(vbo VBO, parameter Param,gpuThread
GPUThread,  dataWaveCompute *dataDev, dataWaveCompute dataHost)
{
   float4 *vertexPos;
   float3 *nptr;
   size_t num_bytes;

   if(Param.device==DEVICE_CPU)
     GPUSimulationAllocationMemory(dataDev,dataHost);

   checkCudaErrors(cudaGraphicsMapResources(1,
     &VBO.cuda_VertexNormal_resource, 0));

   checkCudaErrors(cudaGraphicsResourceGetMappedPointer
    ((void **)&nptr, &num_bytes, VBO.cuda_VertexNormal_resource));
```

```
  checkCudaErrors(cudaGraphicsMapResources(1,
   &VBO.cuda_VertexPos_resource, 0));

  checkCudaErrors(cudaGraphicsResourceGetMappedPointer
    ((void **)&vertexPos, &num_bytes,
           VBO.cuda_VertexPos_resource));

  kernelUpdateVertex<<<GPUThread.grid2,GPUThread.block>>>
       (vertexPos,dataDev->Mesh.VtoE, dataDev->Mesh.VtoS,
        dataDev->Mesh.VertX,  dataDev->Mesh.VertY, dataDev->h,
        dataDev->Mesh.maxVtoEconn,  dataDev->Mesh.NNodes);

  kernelNormalFace<<<GPUThread.grid1,GPUThread.block>>>
       (vertexPos, dataDev->Mesh.NormalFaces, dataDev->Mesh.EtoV,
       dataDev->Mesh.NCells);

  kernelNormalVektor<<<GPUThread.grid2,GPUThread.block>>>
       (vertexPos, nptr, dataDev->Mesh.NormalFaces,
        dataDev->Mesh.VtoE, dataDev->Mesh.maxVtoEconn,
        dataDev->Mesh.NNodes);

  checkCudaErrors(cudaGraphicsUnmapResources(1,
   VBO.cuda_VertexNormal_resource, 0));

  checkCudaErrors(cudaGraphicsUnmapResources(1,
   &VBO.cuda_VertexPos_resource, 0));

  if(Param.device==DEVICE_CPU)  GPUSimulationFreeMemory(dataDev);

}
```

E.12. File renderingVBO.cpp

```
#include "../../header/header.h"


int ttt=0;
double elapsed_time=0.0;

void renderingVBO(GLuint shaderProg, vbo VBO, dataWaveCompute
dataDev, parameter Param)
{
  glEnable(GL_DEPTH_TEST);
  //render dari VBO
  glBindBuffer(GL_ARRAY_BUFFER, VBO.posVertexBuffer);
   glVertexPointer(4, GL_FLOAT, 0, 0);
   glEnableClientState(GL_VERTEX_ARRAY);


 glBindBuffer(GL_ARRAY_BUFFER, VBO.VertexNormalBuffer);
  glClientActiveTexture(GL_TEXTURE0);
  glTexCoordPointer(3, GL_FLOAT, 0, 0);
  glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

```
 glBindBuffer(GL_ARRAY_BUFFER, VBO.signBuffer);
  glClientActiveTexture(GL_TEXTURE1);
  glTexCoordPointer(1, GL_FLOAT, 0, 0);
  glEnableClientState(GL_TEXTURE_COORD_ARRAY);

  glUseProgram(shaderProg);

    GLuint ttimer;
  ttt=(ttt+1)%255;
  ttimer = glGetUniformLocation(shaderProg, "ttimer");
  glUniform1f(ttimer, (ttt/255.0));

  GLuint uniDeepColor, uniShallowColor, uniSkyColor, uniLightDir;

  uniDeepColor = glGetUniformLocation(shaderProg, "deepColor");
  glUniform4f(uniDeepColor, 0.0f, 0.1f, 0.4f, 1.0f);

  uniShallowColor = glGetUniformLocation(shaderProg,
                        "shallowColor");
  glUniform4f(uniShallowColor, 0.1f, 0.3f, 0.3f, 1.0f);

  uniSkyColor = glGetUniformLocation(shaderProg, "skyColor");
  glUniform4f(uniSkyColor, 1.0f, 0.75f, 0.25f, 1.0f);

  uniLightDir = glGetUniformLocation(shaderProg, "lightDir");
  glUniform3f(uniLightDir, 0.0f, 1.0f, 0.0f);

  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VBO.indexBuffer);

  if(Param.style==STYLE_WIRE)
  {
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  }else
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

   glDrawElements(GL_TRIANGLES, dataDev.Mesh.NCells*3+
           dataDev.Mesh.Wall.count*6, GL_UNSIGNED_INT, 0);
   glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);

  glDisableClientState(GL_VERTEX_ARRAY);
   glClientActiveTexture(GL_TEXTURE0);
   glDisableClientState(GL_TEXTURE_COORD_ARRAY);
   glClientActiveTexture(GL_TEXTURE1);
  glDisableClientState(GL_TEXTURE_COORD_ARRAY);
  glDisable (GL_BLEND);

  glUseProgram(0);

}
```

E.13. File zFreeMemory.cu

```
#include "../../header/header.h"
```

```
double GPUSimulationFreeMemory(dataWaveCompute *dataDev)
{

    cudaFree(dataDev->h);
    cudaFree(dataDev->u);
    cudaFree(dataDev->v);
    cudaFree(dataDev->hu);
    cudaFree(dataDev->hv);
    cudaFree(dataDev->z);

    cudaFree(dataDev->hnew);
    cudaFree(dataDev->unew);
    cudaFree(dataDev->vnew);
    cudaFree(dataDev->hunew);
    cudaFree(dataDev->hvnew);

    cudaFree(dataDev->Mesh.L);
    cudaFree(dataDev->Mesh.EtoE);
    cudaFree(dataDev->Mesh.EtoV);
    cudaFree(dataDev->Mesh.VtoE);
    cudaFree(dataDev->Mesh.VtoS);
    cudaFree(dataDev->Mesh.NormalFaces);
    cudaFree(dataDev->Mesh.nx);
    cudaFree(dataDev->Mesh.ny);
    cudaFree(dataDev->Mesh.VertX);
    cudaFree(dataDev->Mesh.VertY);
    cudaFree(dataDev->Mesh.AREA);

    return 1.0;
}
```

E.14. File mesh.cpp

Khusus algoritma untuk menentukan relasi-relasi pada *mesh*, kode program pada bagian ini merupakan modifikasi dan pengembangan dari program milik Tim Warburton, *www.caam.rice.edu/~caam452*.

```
#include "mesh.h"


int imax(int *v, int L){
   int res = v[0];
   int i;
   for(i=1;i<L;++i)
    res=max(res,v[i]);
   return res;
}

int *ivector(int L){
   int *v=(int*) calloc(L,sizeof(int));
   //int *v=malloc(L*sizeof(int));
   return v;
```

```
}

double *dvector(int L){
   double *v=(double*) calloc(L,sizeof(double));
   return v;
}

void dvectorprint(char *message, double *d, int L){
   int i;
   fprintf(stdout, "%s ---start---\n",message);
   for(i=0;i<L;++i){
    fprintf(stdout,"%d: %lf\n",i,d[i]);
   }

   fprintf(stdout, "%s ---end---\n",message);
}
```

```
int readMeshFile(mesh *Mesh, char *title)
{
   int i;

   Mesh->NFaces=3;


// Baca jumlah simpul dan jumlah elemen
   FILE *Fid; //*fp
   char FileName[BUFSIZ], buf[BUFSIZ]; //, fname[BUFSIZ];

   sprintf(FileName, "%s",title);
   if(!(Fid=fopen(FileName,"r"))){
    fprintf(stderr, "Could not load mesh file: %s\n", FileName);
    exit(-1);
   }

//Baca format baris kosong
   for(i=0;i<6;i++){
    fgets(buf, BUFSIZ, Fid);
   }

   fgets(buf, BUFSIZ, Fid);
   sscanf(buf,"%d%d%d",&(Mesh->NNodes), &(Mesh->NCells),
    &Mesh->Wall.count);

   for(i=0; i<2;++i){
    fgets(buf, BUFSIZ, Fid);
   }

// Baca simpul dan cari titik X, Y maksimum dan minimum
   Mesh->VertX=dvector(Mesh->NNodes);
   Mesh->VertY=dvector(Mesh->NNodes);

   for(i=0;i<Mesh->NNodes;++i){
    fgets(buf,BUFSIZ,Fid);

    sscanf(buf, "%lf %lf", Mesh->VertX+i, Mesh->VertY+i);
```

```
    if(i>0){
     if(Mesh->Xmin>*(Mesh->VertX+i))Mesh->Xmin=*(Mesh->VertX+i);
     if(Mesh->Xmax>*(Mesh->VertX+i))Mesh->Xmax=*(Mesh->VertX+i);
     if(Mesh->Ymin>*(Mesh->VertY+i))Mesh->Ymin=*(Mesh->VertY+i);
     if(Mesh->Ymax>*(Mesh->VertY+i))Mesh->Ymax=*(Mesh->VertY+i);
    }else
    {
     Mesh->Xmax=Mesh->Xmin=*Mesh->VertX;
     Mesh->Ymax=Mesh->Ymin=*Mesh->VertY;
    }
   }
   for(i=0;i<2;++i){
      fgets(buf,BUFSIZ,Fid);
    }

//Baca elemen, relasi elemen ke verteks
   Mesh->EtoV=ivector(Mesh->NFaces*Mesh->NCells);

   for(int k=0;k<Mesh->NCells;++k){
    Mesh->EtoV[Mesh->NFaces*k+0]=MESH_FACE_NO_NEIGHBOUR;
    Mesh->EtoV[Mesh->NFaces*k+1]=MESH_FACE_NO_NEIGHBOUR;
    Mesh->EtoV[Mesh->NFaces*k+2]=MESH_FACE_NO_NEIGHBOUR;

     fgets(buf,BUFSIZ,Fid);

     sscanf(buf, "%d %d %d",
       Mesh->EtoV+(Mesh->NFaces*k+0),
       Mesh->EtoV+(Mesh->NFaces*k+1),
       Mesh->EtoV+(Mesh->NFaces*k+2));
   }

//Baca dinding
   Mesh->Wall.Point1=new float3[Mesh->Wall.count];
   Mesh->Wall.Point2=new float3[Mesh->Wall.count];

    fgets(buf,BUFSIZ,Fid);
    fgets(buf,BUFSIZ,Fid);
//   normalWall=  new float3[Mesh->Wall.count*2*3];

    for(int nw=0;nw<Mesh->Wall.count;nw++)
    {
     float wallHeight;
     fgets(buf,BUFSIZ,Fid);
     sscanf(buf, "%f %f %f %f %f",
         &(Mesh->Wall.Point1+nw)->x,
         &(Mesh->Wall.Point1+nw)->y,
         &(Mesh->Wall.Point2+nw)->x,
         &(Mesh->Wall.Point2+nw)->y,
         &wallHeight);
     Mesh->Wall.Point1[nw].z=wallHeight;
     Mesh->Wall.Point2[nw].z=wallHeight;
    }

   /* now close .neu file */
   fclose(Fid);
```

```
   return 1;
}
```

```
void setEtoV(mesh *Mesh, int *Nelmts)
{
   int va,vb,vc;


   // printf("\nWall Count1 = %d", Wall.count  );
   for(int k=0;k<Mesh->NCells ;++k){
    va=Mesh->EtoV[Mesh->NFaces*k+0];
    vb=Mesh->EtoV[Mesh->NFaces*k+1];
    vc=Mesh->EtoV[Mesh->NFaces*k+2];

    Nelmts[va]=Nelmts[va]+1;
    Nelmts[vb]=Nelmts[vb]+1;
    Nelmts[vc]=Nelmts[vc]+1;
   }

   Mesh->maxVtoEconn=imax(Nelmts,Mesh->NNodes)+1;

}
```

```
void   setVtoE_VtoS(mesh *Mesh, int *Nelmts)
{
   int va,vb,vc;

   /* reset Nelmts per node counter */
   Mesh->VtoE = ivector(Mesh->maxVtoEconn*Mesh->NNodes);
   Mesh->VtoS = ivector(Mesh->maxVtoEconn*Mesh->NNodes);
   memset(Mesh->VtoE,-1,
         sizeof(int)*Mesh->maxVtoEconn*Mesh->NNodes);
   memset(Mesh->VtoS,-1,
         sizeof(int)*Mesh->maxVtoEconn*Mesh->NNodes);

   for(int i=0;i<Mesh->NNodes;i++)Nelmts[i]=0;

   /* invert umElmtToNode map */
   for(int k=0; k<Mesh->NCells; ++k){

    va=Mesh->EtoV[Mesh->NFaces*k+0];
    vb=Mesh->EtoV[Mesh->NFaces*k+1];
    vc=Mesh->EtoV[Mesh->NFaces*k+2];
    Mesh->VtoE[Mesh->maxVtoEconn*va+Nelmts[va]]=k;
    Mesh->VtoE[Mesh->maxVtoEconn*vb+Nelmts[vb]]=k;
    Mesh->VtoE[Mesh->maxVtoEconn*vc+Nelmts[vc]]=k;

    Mesh->VtoS[Mesh->maxVtoEconn*va+Nelmts[va]]=Mesh->NFaces*k+0;
    Mesh->VtoS[Mesh->maxVtoEconn*vb+Nelmts[vb]]=Mesh->NFaces*k+1;
    Mesh->VtoS[Mesh->maxVtoEconn*vc+Nelmts[vc]]=Mesh->NFaces*k+2;

    Nelmts[va]=Nelmts[va]+1;
```

```
    Nelmts[vb]=Nelmts[vb]+1;
    Nelmts[vc]=Nelmts[vc]+1;

   }

}
```

```
void setEtoE(mesh *Mesh, int *Nelmts)
{
   int i,j,k;
   int Nbc, *bcelements;
   int va,vb,vc, Nva, Nvc, Nvb, eida, eidb,eidc;


   /* need to create umElmtToElmt */
   Mesh->EtoE=ivector(Mesh->NFaces*Mesh->NCells);
   for(k=0;k<Mesh->NCells;++k)
   {
    va=Mesh->EtoV[Mesh->NFaces*k+0];
    vb=Mesh->EtoV[Mesh->NFaces*k+1];
    vc=Mesh->EtoV[Mesh->NFaces*k+2];

    Nva=Nelmts[va];
    Nvb=Nelmts[vb];
    Nvc=Nelmts[vc];

    Mesh->EtoE[Mesh->NFaces*k+0]=-1;

    for(i=0; i<Nva;++i){
     eida=Mesh->VtoE[Mesh->maxVtoEconn*va+i];
     if(eida!=k){
        for(j=0;j<Nvb;++j){
         eidb=Mesh->VtoE[Mesh->maxVtoEconn*vb+j];
         if(eida==eidb){
             Mesh->EtoE[Mesh->NFaces*k+0]=eida;
         }
        }

     }
    }

    Mesh->EtoE[Mesh->NFaces*k+1]=-1;
    for(i=0; i<Nvb;++i){
     eidb=Mesh->VtoE[Mesh->maxVtoEconn*vb+i];
     if(eidb!=k){
        for(j=0;j<Nvc;++j){
         eidc=Mesh->VtoE[Mesh->maxVtoEconn*vc+j];
         if(eidb==eidc){
             Mesh->EtoE[Mesh->NFaces*k+1]=eidb;
         }
        }

     }
    }
```

```
    Mesh->EtoE[Mesh->NFaces*k+2]=-1;
    for(i=0; i<Nva;++i){
     eida=Mesh->VtoE[Mesh->maxVtoEconn*va+i];
     if(eida!=k){
        for(j=0;j<Nvc;++j){
         eidc=Mesh->VtoE[Mesh->maxVtoEconn*vc+j];
         if(eida==eidc){
             Mesh->EtoE[Mesh->NFaces*k+2]=eida;
         }
        }

     }
    }
   }


   /* find elements sitting at boundary on the inflow */
   Nbc=0;

   for(k=0;k<Mesh->NCells;++k){
    for(i=0;i<Mesh->NFaces;++i){
     if(Mesh->EtoE[Mesh->NFaces*k+i]==-1){
        Nbc=Nbc+1;
     }
    }
   }

   bcelements=ivector(Nbc);
   Nbc=0;


   for(k=0;k<Mesh->NCells;++k){

    for(i=0;i<Mesh->NFaces;++i){
     if(Mesh->EtoE[Mesh->NFaces*k+i]==MESH_FACE_NO_NEIGHBOUR){
        bcelements[Nbc]=k;
        Nbc=Nbc+1;
        Mesh->EtoE[Mesh->NFaces*k+i]=k;

     }
    }
   }
}
```

```
void   setMeshRelation(mesh *Mesh)
{
   //harus urut
   int *Nelmts= ivector(Mesh->NNodes);
   setEtoV(Mesh, Nelmts);
   setVtoE_VtoS(Mesh, Nelmts);
   setEtoE(Mesh, Nelmts);

}
```

```
void setMeshInformation(mesh *Mesh)
{

   double dx, dy;
   double a,b,c,s;

   Mesh->nx=dvector(Mesh->NCells*3);
   Mesh->ny=dvector(Mesh->NCells*3);
   Mesh->L=dvector(Mesh->NCells*3);
   Mesh->AREA=dvector(Mesh->NCells);

   //Mesh->z=dvector(Mesh->NCells);

   Mesh->X=dvector(Mesh->NCells*3);
   Mesh->Y=dvector(Mesh->NCells*3);
   Mesh->CX=dvector(Mesh->NCells);
   Mesh->CY=dvector(Mesh->NCells);
   /*vertex location for each vertex of each element */

   for(int k=0; k<Mesh->NCells; ++k){
    for(int i=0;i<Mesh->NFaces;++i){
     Mesh->X[Mesh->NFaces*k+i]=Mesh->VertX[Mesh->EtoV[Mesh-
>NFaces*k+i]];
     Mesh->Y[Mesh->NFaces*k+i]=Mesh->VertY[Mesh->EtoV[Mesh-
>NFaces*k+i]];

    }

    for(int i=0;i<Mesh->NFaces;++i){
     /*compute edge lengths*/
     dx=(Mesh->X[Mesh->NFaces*k+(i+1)%Mesh->NFaces]-Mesh->X[Mesh-
>NFaces*k+i]);
     dy=(Mesh->Y[Mesh->NFaces*k+(i+1)%Mesh->NFaces]-Mesh->Y[Mesh-
>NFaces*k+i]);
     Mesh->L[Mesh->NFaces*k+i]=sqrt(dx*dx+dy*dy);

     Mesh->nx[Mesh->NFaces*k+i]=dy;
     Mesh->ny[Mesh->NFaces*k+i]=-dx;

     Mesh->nx[Mesh->NFaces*k+i]/=Mesh->L[Mesh->NFaces*k+i];
     Mesh->ny[Mesh->NFaces*k+i]/=Mesh->L[Mesh->NFaces*k+i];
    }
    a=Mesh->L[Mesh->NFaces*k+0];
    b=Mesh->L[Mesh->NFaces*k+0];
    c=Mesh->L[Mesh->NFaces*k+0];
    s=(1/2.)*(a+b+c);

    Mesh->AREA[k]=sqrt(s*(s-a)*(s-b)*(s-c));

    /* cell center coordinates */
    Mesh->CX[k]= (1./(double)3)*(Mesh->X[3*k+0]+Mesh->X[3*k+1] +
Mesh->X[3*k +2]);
    Mesh->CY[k]= (1./(double)3)*(Mesh->Y[3*k+0]+Mesh->Y[3*k+1] +
Mesh->Y[3*k +2]);

   }
```

```
}
```

E.15. File mesh.h

```
#ifndef MESH_H_
#define MESH_H_
#include <vector_types.h>
#include <stdio.h>
#define MESH_FACE_NO_NEIGHBOUR -1

typedef struct
{
   float3 *Point1, *Point2;
   float bold;
   int count;

}wall;


typedef struct{
   int *VtoE;
   int *EtoE;
   int *EtoV;
   int *VtoS;

   int NCells;
   int NNodes;
   int NFaces;
   double *VertX, *VertY;
   double Xmin, Ymin, Xmax, Ymax;
   int maxVtoEconn;
   double *AREA, *L, *nx, *ny, *X, *Y, *CX, *CY;
   float3 *NormalFaces;
   wall Wall;
}mesh;


int imax(int *v, int L);
int *ivector(int L);

double *dvector(int L);

void dvectorprint(char *message, double *d, int L);

int  readMeshFile(mesh *Mesh, char *title);

void setEtoV(mesh *Mesh, int *Nelmts);

void   setVtoE_VtoS(mesh *Mesh, int *Nelmts);

void setEtoE(mesh *Mesh, int *Nelmts);

void   setMeshRelation(mesh *Mesh);
```

```
void   setMeshInformation(mesh *Mesh);

#endif
```

E.16. File vbo.cu

```
#include "vbo.h"

void createVBO2(GLuint *vbo, int size)
{
    glGenBuffers(1, vbo);
    glBindBuffer(GL_ARRAY_BUFFER, *vbo);
    glBufferData(GL_ARRAY_BUFFER, size, 0, GL_DYNAMIC_DRAW);
    glBindBuffer(GL_ARRAY_BUFFER, 0);

    SDK_CHECK_ERROR_GL();
}


void deleteVBO2(GLuint *vbo)
{
    glDeleteBuffers(1, vbo);
    *vbo = 0;
}
```

E.17. File vbo.h

```
#ifndef VBO_H_
#define VBO_H_

#include <GL/glew.h>

#include <cuda_gl_interop.h>
#include <helper_functions.h>    // includes cuda.h and
cuda_runtime_api.h


// CUDA helper functions
#include <helper_cuda.h>         // helper functions for CUDA
error check
#include <helper_cuda_gl.h>

void createVBO2(GLuint *vbo, int size);
void deleteVBO2(GLuint *vbo);


#endif /* VBO_H_ */
```

E.18. File Detail.cu

```
#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif
```

```
#include "header/header.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "user defined lib/vbo.h"
#include <timer.h>
#include <numeric>
#include <iostream>

#define REFRESH_DELAY     10 //ms

const char *Judul
   = "SIMULASI GELOMBANG AIR";

// Ukuran jendela layar
unsigned int window_width  = 2000;
unsigned int window_height = 1000;

// kontrol pada mouse
int mouse_old_x, mouse_old_y;
int mouse_buttons = 0;
float rotate_x = 0.0, rotate_y = 0.0;
float translate_z = -3.0;
float translate_y = 0.0;
float translate_x = 0.0;
float translate_zz=0.0;

//variabel untuk keperluan analisis
//dan pencatatan waktu komputasi
parameter globalParam;
dataAnalysis Analis;

//Data-data komputasi dan visualisasi
GLuint shaderProg;
vbo VBO;
dataWaveCompute dataHost, dataDev;

//Variabel alokasi GPU
gpuThread GPUThread;
int STATUS=PAUSE;
```

```
void mainFlow()
{
   InitMesh(&dataHost.Mesh, "test.txt");
   InitDataComputation(&dataHost,globalParam.dt);
   if(globalParam.device==DEVICE_GPU)
   {
    GPUSimulationAllocationMemory(&dataDev, dataHost);
    threadAllocation(globalParam.threadPerBlock,
             &GPUThread,dataDev);
   }

    initVBO(&VBO,dataHost.Mesh);
```

```
    glutMainLoop();
}
```

```
void display()
{

    if(STATUS==RUNNING)
    {
     Analis.frame++;
     if(globalParam.device==DEVICE_GPU)
     {
        gpuTimingStartRec(&Analis.gCoreTime);
        computeGPU(globalParam.iteration,GPUThread,&dataDev);
        gpuTimingStopRec(&Analis.gCoreTime,
             Analis.gCoreTime.elapsed_time);
        Analis.simulationTime+=dataDev.dt*globalParam.iteration;

     }else
     {
        cpuTimingStartRec(&Analis.cCoreTime);
        computeCPU(globalParam.iteration, &dataHost);
        cpuTimingStopRec(&Analis.cCoreTime,
             Analis.cCoreTime.elapsed_time);
        Analis.simulationTime+=dataHost.dt*globalParam.iteration;
     }
    }

    if(!(globalParam.style==STYLE_NOVISUALIZATION))
    {
        computeVisualization(VBO,globalParam,
             GPUThread,&dataDev,dataHost);

//kontrol layar openGL
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glTranslatef(0.0, 0.0, translate_z);//-3.25);
        glRotatef(rotate_x+5, 1.0, 0.0, 0.0);
        glRotatef(rotate_y -5, 0.0, 1.0, 0.0);
        glTranslatef(-0.5 + translate_x, -0.5, 0.75+translate_y);

        renderingVBO(shaderProg, VBO,dataDev,globalParam);
    }

 if(STATUS==RUNNING)
    if(timesUp(globalParam.endtime,Analis.simulationTime) ||
        frameOver(globalParam.targetFrame,Analis.frame))
      {
        cpuTimingStopRec(&Analis.cVisualTime,
        Analis.cVisualTime.elapsed_time);
        STATUS=PAUSE;
      }
    glutSwapBuffers();
}
```

```
int attachShader(GLuint prg, GLenum type, const char *name)
{
    GLuint shader;
    FILE *fp;
    int size, compiled;
    char *src;

    fp = fopen(name, "rb");

    if (!fp)
    {
        return 0;
    }

    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    src = (char *)malloc(size);

    fseek(fp, 0, SEEK_SET);
    fread(src, sizeof(char), size, fp);
    fclose(fp);

    shader = glCreateShader(type);
    glShaderSource(shader, 1, (const char **)&src,
       (const GLint *)&size);
    glCompileShader(shader);
    glGetShaderiv(shader, GL_COMPILE_STATUS,
       (GLint *)&compiled);

    if (!compiled)
    {
        char log[2048];
        int len;

        glGetShaderInfoLog(shader, 2048,
            (GLsizei *)&len, log);
        printf("Info log: %s\n", log);
        glDeleteShader(shader);
        return 0;
    }

    free(src);
    glAttachShader(prg, shader);
    glDeleteShader(shader);

    return 1;
}
```

```
GLuint loadGLSLProgram(const char *vertFileName,
    const char *fragFileName)
{
    GLint linked;
```

```
    GLuint program;

    program = glCreateProgram();

    if (!attachShader(program, GL_VERTEX_SHADER, vertFileName))
    {
        glDeleteProgram(program);
        fprintf(stderr,
            "kesalahan attach vertek %s\n", vertFileName);
        return 0;
    }

    if (!attachShader(program, GL_FRAGMENT_SHADER, fragFileName))
    {
        glDeleteProgram(program);
        fprintf(stderr, "Ckesalahan attach fragment %s\n",
            fragFileName);

        return 0;
    }

    glLinkProgram(program);
    glGetProgramiv(program, GL_LINK_STATUS, &linked);

    if (!linked)
    {
        glDeleteProgram(program);
        char temp[256];
        glGetProgramInfoLog(program, 256, 0, temp);
        fprintf(stderr, "Failed to link program: %s\n", temp);
        return 0;
    }

    return program;
}
```

```
double cpuSecond(){
   struct timeval tp;
   gettimeofday(&tp,NULL);
   return ((double)tp.tv_sec
     + (double)tp.tv_usec*1.e-6);
}

void gpuTimingStartRec(gpuTiming *GPUTiming)
{

    cudaEventCreate( &GPUTiming->start) ;
     cudaEventCreate( &GPUTiming->stop) ;
     cudaEventRecord( GPUTiming->start, 0 ) ;
}
void gpuTimingStopRec(gpuTiming *GPUTiming,
    float prevElapsed)
{
    cudaEventRecord( GPUTiming->stop, 0 ) ;
```

```
    cudaEventSynchronize( GPUTiming->stop ) ;
    cudaEventElapsedTime( &GPUTiming->elapsed_time,
        GPUTiming->start, GPUTiming->stop ) ;
    cudaEventDestroy( GPUTiming->start ) ;
    cudaEventDestroy( GPUTiming->stop ) ;
    GPUTiming->elapsed_time/=1000;
    GPUTiming->elapsed_time+=prevElapsed;

}

void cpuTimingStartRec(cpuTiming *CPUTiming)
{
   CPUTiming->start=cpuSecond();

}
void cpuTimingStopRec(cpuTiming *CPUTiming,float prevElapsed)
{
   CPUTiming->stop=cpuSecond();
   CPUTiming->elapsed_time=CPUTiming->stop-CPUTiming->start
     +prevElapsed;
}

void initParam(parameter *Param)
{
   Param->device=DEVICE_GPU;
   Param->iteration=20;
   Param->style=STYLE_SURFACE;
   Param->endtime=100;
   Param->threadPerBlock=128;
   Param->targetFrame=-1;
   Param->dt=0.00075;

}

void initAnalis(dataAnalysis *Analis)
{
   Analis->frame=0;
   Analis->cVisualTime.elapsed_time=0;
   Analis->cCoreTime.elapsed_time=0;
   Analis->gCoreTime.elapsed_time=0;

   Analis->cVisualTime.stop=0;
   Analis->cCoreTime.stop=0;
   Analis->gCoreTime.stop=0;

   Analis->cVisualTime.start=0;
   Analis->cCoreTime.start=0;
   Analis->gCoreTime.start=0;
   Analis->simulationTime=0;
}


void showAnalis(parameter P, dataAnalysis A)
{
   printf("\n\n......Data Analisis.....\n");
   printf("\nThreadPerBlock\t\t: %d",P.threadPerBlock);
```

```
   printf("\nIterationPerFrame\t: %d",P.iteration);
   printf("\nFrame\t\t\t: %d",A.frame);
   printf("\nTotal Iteration\t\t: %d",
      P.iteration*(A.frame));
   printf("\nElapsed Time CPU\t: %0.3lf",
      A.cCoreTime.elapsed_time);
   printf("\nElapsed Time GPU\t: %0.3lf",
      A.gCoreTime.elapsed_time);
   if(P.style==STYLE_NOVISUALIZATION)
    A.cVisualTime.elapsed_time=0;
   printf("\nVisualization Time\t: %0.3lf",
      A.cVisualTime.elapsed_time);
   printf("\nSimulation Real Time\t: %0.3lf",
      A.simulationTime);
   printf("\n");
}

int timesUp( double endTime,double elapsedSimulation)
{
   return (elapsedSimulation>=endTime && endTime>0);
}

int frameOver( int targetFrame,int frame)
{  glVertex3f(1.02, 0.0,0.37);
   return (frame>=targetFrame && targetFrame>0);
}
```

```
void timerEvent(int value)
{
    if (glutGetWindow())
    {
        glutPostRedisplay();
        glutTimerFunc(REFRESH_DELAY, timerEvent,0);
    }
}

void cleanup()
{
    deleteVBO2(&VBO.posVertexBuffer);
    deleteVBO2(&VBO.VertexNormalBuffer);
    deleteVBO2(&VBO.signBuffer);

    GPUSimulationFreeMemory(&dataDev);
    showAnalis(globalParam,Analis);
    glutDestroyWindow(glutGetWindow());

    cudaDeviceReset();
}


void keyboard(unsigned char key, int /*x*/, int /*y*/)
{
    switch (key)
    {
      case 13:
```

```
        if(STATUS==PAUSE)
        {
            STATUS=RUNNING;
            cpuTimingStartRec(&Analis.cVisualTime);
        }else
        {
            STATUS=PAUSE;
            cpuTimingStopRec(&Analis.cVisualTime,
                  Analis.cVisualTime.elapsed_time);
        }
      break;
      case (27) :
        if(STATUS==RUNNING)
            cpuTimingStopRec(&Analis.cVisualTime,
                  Analis.cVisualTime.elapsed_time);
            #if defined(__APPLE__) || defined(MACOSX)
                exit(EXIT_SUCCESS);
            #else
                glutLeaveMainLoop();
                return;
            #endif
    }
}

////////////////////////////////////////////////////////////////////////
//////////////
//! Mouse event handlers
////////////////////////////////////////////////////////////////////////
//////////////
void mouse(int button, int state, int x, int y)
{
   if (state == GLUT_DOWN && button==GLUT_MIDDLE_BUTTON)
   {
    mouse_buttons=2;
   }
   else if (state == GLUT_DOWN)
    {
        mouse_buttons |= 1<<button;
    }
    else if (state == GLUT_UP)
    {
        mouse_buttons = 0;
    }

    mouse_old_x = x;
    mouse_old_y = y;
}

void motion(int x, int y)
{
    float dx, dy;
    dx = (float)(x - mouse_old_x);
    dy = (float)(y - mouse_old_y);
    if (mouse_buttons & 1)
    {
        rotate_x += dy * 0.2f;
```

```
        rotate_y += dx * 0.2f;
    }
    else if (mouse_buttons & 4)
    {
        translate_z += dy * 0.01f;
    }
    else if (mouse_buttons ==2)
    {
     translate_y += dy*0.01f;
     translate_x += dx*0.01f;

    }
    mouse_old_x = x;
    mouse_old_y = y;
}
```

```
int main(int argc, char **argv)
{
    char *ref_string= NULL;

    initParam(&globalParam);
    initAnalis(&Analis);

#if defined(__linux__)
    setenv ("DISPLAY", ":0", 0);
#endif

    printf("%s starting...\n", Judul);

    if (argc > 1)
    {


        if (checkCmdLineFlag(argc, (const char **)argv, "frame"))
        {

            getCmdLineArgumentString(argc, (const char **)argv,
                        "frame", (char **)&ref_string);
            sscanf(ref_string,"%d",&globalParam.targetFrame);

        }

        if (checkCmdLineFlag(argc, (const char **)argv, "file"))
        {

            getCmdLineArgumentString(argc, (const char **)argv,
                        "file", (char **)&ref_string);
            printf("\nNama File : %s", ref_string);
            globalParam.sourceFile=ref_string;
        }
        if (checkCmdLineFlag(argc, (const char **)argv, "thread"))
    {

     getCmdLineArgumentString(argc, (const char **)argv,
```

```
        "thread", (char **)&ref_string);
     printf("\nNAlokasi Thread : %s", ref_string);
     sscanf(ref_string,"%d",&globalParam.threadPerBlock);
     printf("\nintThread : %d", globalParam.threadPerBlock);


    }
        if (checkCmdLineFlag(argc, (const char **)argv,
"endtime"))
    {

     getCmdLineArgumentString(argc, (const char **)argv,
        "endtime", (char **)&ref_string);
     printf("\nEnd Time : %s", ref_string);
     sscanf(ref_string,"%lf",&globalParam.endtime);

    }
        if (checkCmdLineFlag(argc, (const char **)argv, "dt"))
    {

     getCmdLineArgumentString(argc, (const char **)argv,
        "dt", (char **)&ref_string);
     printf("\nTime Step : %s", ref_string);
     sscanf(ref_string,"%lf",&globalParam.dt);

    }
        if (checkCmdLineFlag(argc, (const char **)argv, "style"))
    {

     getCmdLineArgumentString(argc, (const char **)argv,
        "style", (char **)&ref_string);
     printf("Visual Style : %s", ref_string);
     if(strcmp(ref_string,"novisualization")==0)
     {
       globalParam.style=STYLE_NOVISUALIZATION;

     }else if(strcmp(ref_string,"surface")==0)
     {
       globalParam.style=STYLE_SURFACE;

     }else if(strcmp(ref_string,"hsv")==0)
     {
       globalParam.style=STYLE_COLORHSV;
     }
     else if(strcmp(ref_string,"wire")==0)
      {
            globalParam.style=STYLE_WIRE;
      }
     else
     {
       int tempInt;
       sscanf(ref_string,"%d",&tempInt);
       if(tempInt<3)
       {
        globalParam.style=tempInt;
       }
```

```
     }


    }
        if (checkCmdLineFlag(argc, (const char **)argv, "device"))
      {

       getCmdLineArgumentString(argc, (const char **)argv,
           "device", (char **)&ref_string);
       printf("\nPilihan Device: %s", ref_string);
       if(strcmp(ref_string,"cpu")==0)
       {
        globalParam.device=DEVICE_CPU;
       }else if (strcmp(ref_string,"gpu")==0)
       {
        globalParam.device=DEVICE_GPU;
       }
      }
        if (checkCmdLineFlag(argc, (const char **)argv,
            "iteration"))
      {

       getCmdLineArgumentString(argc, (const char **)argv,
           "iteration", (char **)&ref_string);
       printf("\nIterasi : %s", ref_string);
       int tempInt;

     sscanf(ref_string,"%d",&tempInt);
     if(tempInt>0)
     {
       globalParam.iteration=tempInt;
     }
      }
        if (checkCmdLineFlag(argc, (const char **)argv, "out"))
      {

       getCmdLineArgumentString(argc, (const char **)argv,
           "out", (char **)&ref_string);
       printf("\nOut file : %s", ref_string);
       if(strcmp(ref_string,"")!=0)
       {
        globalParam.outFile=ref_string;
       }
      }
    }

    if(globalParam.style==STYLE_NOVISUALIZATION)
    {
     STATUS=RUNNING;
     printf("\nNO Visualization");
     printf("\nProcess...");

    }
    runTest(argc, argv, NULL);//ref_string
}
```

```
bool initGL(int *argc, char **argv)
{
    glutInit(argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Cuda GL Interop (VBO)");
    glutFullScreen();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutMotionFunc(motion);
    glutTimerFunc(REFRESH_DELAY, timerEvent,0);

    char* vertShaderPath = sdkFindFilePath("test.vert", argv[0]);
    char* fragShaderPath = sdkFindFilePath("test.frag", argv[0]);
    // initialize necessary OpenGL extensions
    glewInit();

    if (! glewIsSupported("GL_VERSION_2_0 "))
    {
        fprintf(stderr, "ERROR: Support for necessary OpenGL
extensions missing.");
        fflush(stderr);
        return false;
    }

    // default initialization
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glDisable(GL_DEPTH_TEST);
   // viewport
    glViewport(0, 0, window_width, window_height);

    // projection
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60.0, (GLfloat)window_width / (GLfloat)
       window_height, 0.1, 10.0);
    shaderProg = loadGLSLProgram(vertShaderPath, fragShaderPath);
    SDK_CHECK_ERROR_GL();

    return true;
}
```

```
bool runTest(int argc, char **argv, char *ref_file)
{
    if (ref_file != NULL)
    {   printf("\nTEST");
        int devID = findCudaDevice(argc, (const char **)argv);
        cudaDeviceReset();
    }
    else    glVertex3f(1.02, 0.0,0.37);
    {
        if (false == initGL(&argc, argv))
        {
            return false;
        }
```

```
        if (checkCmdLineFlag(argc, (const char **)argv, "device"))
        {
            if (gpuGLDeviceInit(argc, (const char **)argv) == -1)
           {
               return false;
            }
        }
        else
        {
            cudaGLSetGLDevice(gpuGetMaxGflopsDeviceId());
        }
       // register callbacks
        glutDisplayFunc(display);
        glutKeyboardFunc(keyboard);
        glutMouseFunc(mouse);
        glutMotionFunc(motion);
#if defined (__APPLE__) || defined(MACOSX)
        atexit(cleanup);
#else
        glutCloseFunc(cleanup);
#endif
        mainFlow();
    }
    return true;
}

#ifdef _WIN32
#ifndef FOPEN
#define FOPEN(fHandle,filename,mode) fopen_s(&fHandle, filename,
                                                    mode)
#endif
#else
#ifndef FOPEN
#define FOPEN(fHandle,filename,mode) (fHandle = fopen(filename,
                                                  mode))
#endif
#endif

void sdkDumpBin2(void *data, unsigned int bytes, const char
*filename)
{
    printf("sdkDumpBin: <%s>\n", filename);
    FILE *fp;
    FOPEN(fp, filename, "wb");
    fwrite(data, bytes, 1, fp);
    fflush(fp);
    fclose(fp);
}
```

E.19. GLSL Vertex Shader “test.vert”

```
// GLSL vertex shader
```

```
varying vec3 worldSpaceNormal;
varying vec3 eyeSpaceNormal;
varying vec3 eyeSpacePos;
varying float sgn;
varying float height;
varying float idx;

void main()
{
   float3 normal     = gl_MultiTexCoord0.xyz;
   float sign        = gl_MultiTexCoord1.x;
   worldSpaceNormal  = normalize(vec3(normal.x,normal.y,normal.z));
   sgn               = sign;

   vec4 pos;
   pos    = vec4(gl_Vertex.x, gl_Vertex.y, gl_Vertex.z, 1.0);
   height = gl_Vertex.y;
   idx    = gl_VertexID;
   gl_Position      =  gl_ModelViewProjectionMatrix * pos;
   eyeSpacePos      = (gl_ModelViewMatrix * pos).xyz;
   eyeSpaceNormal   = (gl_NormalMatrix * worldSpaceNormal).xyz;
 }
```

E.20. GLSL Fragment Shader “test.frag”

```
// GLSL fragment shader

varying vec3 eyeSpacePos;
varying vec3 worldSpaceNormal;
varying vec3 eyeSpaceNormal;
varying float sgn;
varying float height;
varying float idx;

uniform float ttimer;
uniform vec4 deepColor;
uniform vec4 shallowColor;
uniform vec4 skyColor;
uniform vec3 lightDir;
varying vec3 sgn2;

void main()
{

    vec3 eyeVector             = normalize(eyeSpacePos);
    vec3 eyeSpaceNormalVector  = normalize(eyeSpaceNormal);
    vec3 worldSpaceNormalVector = normalize(worldSpaceNormal);
    float sig        = sgn;
    float facing     = max(0.0, dot(eyeSpaceNormalVector, -
                                    eyeVector));
    float fresnel    = pow(1.0 - facing, 5.0);
    float diffuse    = max(0.0, dot(worldSpaceNormalVector,
                                    lightDir));
```

```
    vec4 waterColor = mix(shallowColor, deepColor, facing);
    float zval=height;
    float myColor[3];

   if (zval < 0.2)
    { myColor[0]=0.5*(1.0-zval/0.2);myColor[1]=0.0;
       myColor[2]=0.5+(0.5*zval/0.2);}

  zval*=4.2;

  if ((zval >= 0.2) && (zval < 0.40))     // blue to cyan ramp
   { myColor[0]= 0.0; myColor[1]= (zval-0.2)*5.0;
      myColor[2] = 1.0; }
  if ((zval >= 0.40) && (zval < 0.6))     // cyan to green ramp
   { myColor[0]= 0.0; myColor[1]= 1.0;
       myColor[2] = (0.6-zval)*5.0; }
  if ((zval >= 0.6) && (zval < 0.8))      // green to yellow ramp
   { myColor[0]= (zval-0.6)*5.0; myColor[1]= 1.0;
      myColor[2] = 0.0; }
  if (zval >= 0.8)  // yellow to red ramp
   { myColor[0]= 1.0; myColor[1]= (1.0-zval)*5.0;
      myColor[2]= 0.0; }

    vec4 color1 = skyColor*fresnel;
    vec4 color2 = waterColor*diffuse ;
    vec4 color;

  color=color1+color2;

  /*   // Khusus untuk DamBreak_Validasi, warna dinding pilar
  if(idx>19742.0)
   color=vec4(0.8,0.5,0.5,0.5);

  if(idx>19766.0)
   color=vec4(0.2,0.1,0.2,0.5);
  if(idx>19772.0)
   color=vec4(0.3,0.1,0.2,0.5);
  */

  gl_FragColor = color;

}
```