

Jun 05, 15 10:16

main.c

Page 1/1

```

#include <time.h>
#include "phiADT.h"

// Local Method Declarations
void parse_cmdLineArgs(int argc, char *argv[]);

// Global variables
char *fileToRead; // name of the file to read
char *prefix;      // prefix for the output file

/*
 * Main method requires two command line arguments
 * Usage: program <file_to_process> <output_prefix>
 */
int main (int argc, char *argv[]) {

    time_t start, stop;
    time(&start);

    parse_cmdLineArgs(argc, argv);

    // open the file as read-only
    FILE *inputFile = fopen( fileToRead, "r" );
    if ( inputFile == NULL ) exit(1);

    char fileName[255];
    sprintf(fileName, "%s_distfield", prefix);

    Phi *phiFncn = phi_create(inputFile);
    phi_calc_distance_field(phiFncn);
    phi_gen_file(phiFncn, VTI|DAT, fileName);
    phi_destroy(phiFncn);

    fclose(inputFile);

    time(&stop);
    printf("\n-----\n");
    printf("Finished in about %fs\n", difftime(stop, start));
    printf("-----\n\n");

    return EXIT_SUCCESS;
}

/*
 * Method to parse and check the command line
 * arguments passed when running the code.
 * Arguments:
 *     int [in] - total number of arguments passed
 *     char* [] [in] - pointer to char array
 * Returns:
 */
void parse_cmdLineArgs(int argc, char *argv[]) {
    if ( argc != 3 ){
        printf("\nUsage: distance_field <file_to_process> <output_prefix>\n\n");
        exit(EXIT_FAILURE);
    }

    fileToRead = argv[1];
    prefix = argv[2];
}

```

Jun 05, 15 10:18

phiADT.h

Page 1/1

```
/*
 * phiADT.h
 * Abstract Data Type for the Distance Field
 * Preprocessor.
 *
 * Created on: Jun 04, 2015
 * Author: Anup Shrestha
 *
 * Audit Trail:
 * Date:
 * Changes:
 */
#ifndef PHIADT_H_
#define PHIADT_H_

#include <stdio.h>
#include <stdlib.h>
#include "file_writer.h"

// Type Definitions
typedef struct phi_type Phi;

// Method Declarations
Phi *phi_create(FILE *);
void phi_destroy(Phi *);
void phi_calc_distance_field(Phi *);
void phi_gen_file(Phi *, FileOut, char *);

#endif /* PHIADT_H_ */
```

Jun 05, 15 10:41

phi3D.h

Page 1/1

```

/*
 * phi3D.h
 * Main header file for 3D Phi Function.
 *
 * Created on: Jun 04, 2015
 * Author: Anup Shrestha
 *
 * Audit Trail:
 * Date:
 * Changes:
 */

#ifndef PHI3D_H_
#define PHI3D_H_

#include "phiADT.h"
#include "vti_parser.h"
#include <math.h>

/* speed of propagation [Equation (1) in the report] */
#define SPEED 1
#define DEFAULT_BORDER_LOCATION -1
#define DEFAULT_BORDER_DISTANCE INFINITY
#define DEFAULT_INTERIOR_DISTANCE 90000

/*
 * Completing the struct declared in the
 * phiADT.h This definition is for a 3D phi.
 * x, y, z - interior dimensions
 * dx, dy, dz - node spacing
 * F - speed
 * location - location values
 * distance - distance values
 */
struct phi_type {
    int x, y, z;
    double dx, dy, dz, F;
    int * location;
    double * distance;
};

#endif /* PHI3D_H_ */

```

```

Jun 05, 15 10:41      phi3D.c      Page 1/3

/*
 * phi3D.c
 * This is a wrapper file that calls appropriate
 * methods from other different files that:
 * 1) Creates and initializes the phi function
 *    variables
 * 2) Calculates the distance field
 * 3) Writes the data to file
 * 4) Performs cleanup
 *
 * Created on: Jun 04, 2015
 * Author: Anup Shrestha
 *
 * Audit Trail:
 * Date:
 * Changes:
 */
#include "phi3D.h"

// private method declarations
static void adjust_boundary(Phi *p, Grid3D gridEx);
static int linear3dIndex(int x, int y, int z, int max_x, int max_y);

// external method declarations
extern void init_phiFcn(Phi *p, double *d);
extern void free_phiFcn(Phi *p);
extern void calc_distfield(Phi *p, Grid3D g3d);

/*
 * Creates phi function, calls the file parser
 * methods to get the dimensions, location
 * and distance data from the file. Also,
 * allocates memory and initializes variables
 * required by phi function.
 * Arguments:
 * FILE* [in] - pointer to a file
 * Returns:
 * Phi* - pointer to phi function
 */
Phi* phi_create(FILE *vti) {
    Phi *p = (Phi *) malloc(sizeof(Phi));

    // parsing the information about the dimensions
    // and the spacing of the nodes from the file and
    // storing them in an array
    double *dims = (double *) malloc(sizeof(double) * 6);
    vti_get_dimensions(vti, dims);

    // now using the information about the dimensions
    // and the spacing create a phi function by
    // initializing all the attributes and allocating
    // space for the location and distance data
    init_phiFcn(p, dims);

    // the phi function stores the internal dimensions
    // so store the external dimensions in a global
    // Grid3D struct variable for future use.
    // Grid3D and make_grid3D are defined in vti_parser.h
    Grid3D gridEx = make_grid3D(p->x + 2, p->y + 2, p->z + 2);

    // now parse the location and distance data from
    // the file and store it in appropriate arrays
    // inside the phi function.
    vti_get_data(vti, p->location, DEFAULT_BORDER_LOCATION,
                p->distance, DEFAULT_BORDER_DISTANCE, gridEx);

```

```

Jun 05, 15 10:41      phi3D.c      Page 2/3

        return p;
    }

/*
 * Calls the method that deallocates
 * memory allocated for the phi function.
 * Arguments:
 * Phi* [in] - pointer to phi function
 * Returns:
 */
void phi_destroy(Phi *p) {
    free_phiFcn(p);
}

/*
 * Calls the method that calculates the distance
 * field.
 * Arguments:
 * Phi* [in/out] - pointer to phi function
 * Returns:
 */
void phi_calc_distance_field(Phi *p) {
    Grid3D gridEx = make_grid3D(p->x + 2, p->y + 2, p->z + 2);
    calc_distfield(p, gridEx);
}

/*
 * Updates the boundary values of the distance
 * array, then calls the method that writes
 * distance field to different files based on
 * the FileOut parameter
 * Arguments:
 * Phi* [in] - pointer to phi function
 * FileOut [in] - Output file type enumerator
 * char* [in] - name of the output file
 * Returns:
 */
void phi_gen_file(Phi *p, FileOut out, char *fileName) {
    Grid3D gridEx = make_grid3D(p->x + 2, p->y + 2, p->z + 2);
    adjust_boundary(p, gridEx);
    FileGrid fg = make_fileGrid(gridEx.x, gridEx.y, gridEx.z, p->dx, p->dy, p->dz);
    FileType ft = make_fileType(fileName, p->distance, out, fg);

    // generate file(s)
    file_generate(&ft);
}

/*
 * Adjusts the boundary values
 * Arguments:
 * Phi* [in/out] - pointer to phi function
 * Grid3D [in] - dimensions of grid
 * Returns:
 */
static void adjust_boundary(Phi *p, Grid3D gridEx) {
    // before writing to file we need to update the
    // boundary values
    int x, y, z, i, j, k;
    x = gridEx.x;
    y = gridEx.y;
    z = gridEx.z;

```

Jun 05, 15 10:41

phi3D.c

Page 3/3

```

    for(i = 0; i < z; i++){
        for(j = 0; j < y; j++){
            for(k = 0; k < x; k++){
                int I = i, J = j, K = k;
                I = (i == z-1) ? I-1 : (!i) ? I+1 : I;
                J = (j == y-1) ? J-1 : (!j) ? J+1 : J;
                K = (k == x-1) ? K-1 : (!k) ? K+1 : K;
                if( i != I || j != J || k != K) {
                    p->distance[linear3dIndex(k, j, i, x, y)] = p->distance[linear3d
Index(K, J, I, x, y)];
                }
            }
        }
    }
}

/*
 * Convert 3D indexing to 1D indexing
 * Arguments:
 *   int x, y, z [in] - 3D coordinate
 *   int max_x   [in] - size of x-dimension
 *   int max_y   [in] - size of y-dimension
 * Returns:
 */
static int linear3dIndex(int x, int y, int z, int max_x, int max_y) {
    return z * max_y * max_x + y * max_x + x;
}

```

Jun 05, 15 10:41

phi3D_cuda.h

Page 1/1

```

/*
 * phi3D_cuda.h
 * Phi CUDA header file.
 *
 * Created on: Jun 04, 2015
 * Author: Anup Shrestha
 *
 * Audit Trail:
 * Date:
 * Changes:
 */

#ifndef PHI3D_CUDA_H_
#define PHI3D_CUDA_H_

#include "phi3D.h"

/*
 * Structure for storing the cudaExtent and the
 * cudaPitchedPtr for the 3D array memory in
 * device (GPU) for location and distance.
 */
typedef struct {
    cudaExtent loc_ext, dst_ext;
    cudaPitchedPtr loc_dPitchPtr, dst_dPitchPtr;
} Phi3D_d;

/*
 * Structure for storing information used during
 * sweeping to manage internal grid dimensions,
 * sweep directions, position of the node on the
 * array and its offset in each kernel block
 */
typedef struct {
    int level;
    int xDim, yDim, zDim;
    int xOffSet, yOffset;
    int xSweepOff, ySweepOff, zSweepOff;
    double dx, dy, dz;
} SweepInfo;

// Macro for checking CUDA errors following a CUDA launch or API call
#define cudaCheckError() {\
    cudaError_t e = cudaGetLastError();\
    if( e != cudaSuccess ) {\
        printf("\nCuda failure %s:%d: '%s'\n", __FILE__, __LINE__, cudaGetErrorS\
tring(e));\
        exit(EXIT_FAILURE);\
    }\
}

// External Linkage Method Declarations
extern "C" {
    void init_phiFcn(Phi *p, double *d);
    void free_phiFcn(Phi *p);
    void calc_distfield(Phi *p, Grid3D g3d);
}

#endif /* PHI3D_CUDA_H_ */

```

Jun 05, 15 10:42

phi3D_cuda.c

Page 1/8

```

/*
 * phi3D_cuda.cu
 * Phi3D CUDA source file.
 *
 * Parallel 3D implementation of Fast Sweeping Method
 * using CUDA C/C++.
 * Takes the information about locations and IB distances
 * from a VTI file and propagates it. The algorithm
 * implemented for parallel fast sweeping method is from
 * a paper in the Journal of Computational Physics titled
 * "A parallel fast sweeping method for the Eikonal Equation"
 * by Miles Detrixhe, Federic Gibou, and Chohong Min.
 * DOI: http://www.sciencedirect.com/science/article/pii/S002199911200722X
 *
 * Created on: Jun 04, 2015
 * Author: Anup Shrestha
 *
 * Audit Trail:
 * Date:
 * Changes:
 */
#include "phi3D_cuda.h"

static int iDivUp(int a, int b) {
    return ( (a % b) != 0 ) ? (a / b + 1) : (a / b);
}

// private method declarations
static void update_distance(int *loc_d, double *dst_d, Grid3D g3d);
static void fast_sweep(Phi* p, int noOfTimesToSweep, cudaPitchedPtr dst_dPitchPtr);
static void set_distance_negative_inside(Phi3D_d *d_p, Grid3D g3d);
static void cudaMemcpy3D_P2D(Phi3D_d *d_p, double *dst, int *loc);
static void cudaMemcpy3D_D2P(Phi3D_d *d_p, double *dst, int *loc);

// Kernel Declarations
__global__ void update_distance_kernel(int *loc, double *dst, int totalNodes);
__global__ void set_distance_negative_inside_kernel(cudaPitchedPtr dst_dPitchPtr,
    cudaPitchedPtr loc_dPitchPtr, Grid3D g3d);
__global__ void fast_sweep_kernel(cudaPitchedPtr dst_dPitchPtr, SweepInfo s);
__device__ double solve_eikonal(double cur_dist, double minX, double minY, double
    minZ, double dx, double dy, double dz);

/*
 * Initializes all the attributes of the phi function.
 * Also allocates pinned memory for location and
 * distance arrays. This function is called after the
 * input VTI file has been processed, and the dimensions
 * of the grid are known.
 * Arguments:
 *   Phi3D [out] - pointer to phi function
 *   double* [in] - pointer to array of double that
 *                   has the dimension and spacing values
 * Returns:
 */
void init_phiFcn(Phi *p, double *d) {
    p->x = (int) d[0] + 1; p->dx = d[3];
    p->y = (int) d[1] + 1; p->dy = d[4];
    p->z = (int) d[2] + 1; p->dz = d[5];
    p->F = SPEED;

    // allocating pinned memory for the
    // location and distance arrays
    int totalNodes = (p->x + 2) * (p->y + 2) * (p->z + 2);
    size_t l_arr = sizeof(int) * totalNodes;
    size_t d_arr = sizeof(double) * totalNodes;

```

Jun 05, 15 10:42

phi3D_cuda.c

Page 2/8

```

    cudaHostAlloc((void **)&p->location, l_arr, cudaHostAllocMapped);
    cudaCheckError();
    cudaHostAlloc((void **)&p->distance, d_arr, cudaHostAllocMapped);
    cudaCheckError();
}

/*
 * Deallocates memory used by the phi function
 * Arguments:
 *   Phi* [in] - pointer to phi function
 * Returns:
 */
void free_phiFcn(Phi *p) {
    cudaFreeHost(p->location); cudaCheckError();
    cudaFreeHost(p->distance); cudaCheckError();
    free(p);
}

/*
 * This method updates the distance array based
 * on the location values. Then allocates 3D
 * memory on the device, copies the memory from
 * pinned memory to device memory, runs the fast
 * sweeping method, sets the inside distance to
 * negative based on the location values and
 * copies the device memory back to pinned memory.
 * Arguments:
 *   Phi* [in] - pointer to phi function
 *   Grid3D [in] - dimensions of the grid
 * Returns:
 */
void calc_distfield(Phi *p, Grid3D g3d) {
    // get the device pointers to the pinned memory
    int *loc_d; double *dst_d;
    cudaHostGetDevicePointer(&loc_d, p->location, 0); cudaCheckError();
    cudaHostGetDevicePointer(&dst_d, p->distance, 0); cudaCheckError();

    update_distance(loc_d, dst_d, g3d);

    /*
     * Setup for running the fast sweeping method on the
     * device (GPU). For faster performance, we want to
     * allocate memory for the arrays using cudaMalloc3D.
     * Even though the pinned memory has a device pointer
     * and can be accessed directly by kernels, it has not
     * been allocated using cudaMalloc3D, hence it degrades
     * performance. Therefore, we need to:
     * 1) Allocated memory using cudaMalloc3D
     * 2) Copy the pinned memory data to that memory
     * 3) Perform calculations
     * 4) Copy the data back to pinned memory
     * 5) Deallocate device memory
     */

    Phi3D_d* d_p = (Phi3D_d *) malloc (sizeof(Phi3D_d));

    /* (1) */
    d_p->loc_ext = make_cudaExtent(g3d.x * sizeof(int), g3d.y, g3d.z);
    d_p->dst_ext = make_cudaExtent(g3d.x * sizeof(double), g3d.y, g3d.z);

    cudaMalloc3D(&d_p->loc_dPitchPtr, d_p->loc_ext); cudaCheckError();
    cudaMalloc3D(&d_p->dst_dPitchPtr, d_p->dst_ext); cudaCheckError();

    /* (2) */
    cudaMemcpy3D_P2D(d_p, dst_d, loc_d);

```

Jun 05, 15 10:42

phi3D_cuda.c

Page 3/8

```

/* (3) */
fast_sweep(p, 3, d_p->dst_dPitchPtr);

set_distance_negative_inside(d_p, g3d);

/* (4) */
cudaMemcpy3D_D2P(d_p, dst_d, loc_d);

/* (5) */
cudaFree(d_p->loc_dPitchPtr.ptr);
cudaFree(d_p->dst_dPitchPtr.ptr);
free(d_p);
}

/*
 * Sets up the kernel call that updates the distance
 * array based on the location values.
 * Arguments:
 *   int* [in]      - location array in device
 *   double* [in/out] - distance array in device
 *   Grid3D [in]    - full grid dimensions
 * Returns:
 */
static void update_distance(int *loc_d, double *dst_d, Grid3D g3d) {

    // Setup 3D-Grid and 3D-Block for Kernel launch
    // Running 256 threads per block
    dim3 bs(8, 8, 8);
    dim3 gs(iDivUp(g3d.x, bs.x), iDivUp(g3d.y, bs.y), iDivUp(g3d.z, bs.z));

    int totalNodes = g3d.x * g3d.y * g3d.z;
    update_distance_kernel(<<gs, bs>>>(loc_d, dst_d, totalNodes);
    cudaThreadSynchronize(); cudaCheckError();

}

/*
 * Kernel that updates the distance array based
 * on the location values.
 * Arguments:
 *   int* [in]      - location array in device
 *   double* [in/out] - distance array in device
 *   int [in]       - total number of nodes
 */
__global__ void update_distance_kernel(int *loc, double *dst, int totalNodes) {
    int blockIdx = blockIdx.x + blockIdx.y * gridDim.x + gridDim.x * gridDim.
y * blockIdx.z;
    int tid = blockIdx * (blockDim.x * blockDim.y * blockDim.z) +
        (threadIdx.z * (blockDim.x * blockDim.y)) +
        (threadIdx.y * blockDim.x) + threadIdx.x;

    if (tid < totalNodes) {
        int l = loc[tid];
        double d = dst[tid];
        if (l != DEFAULT_BORDER_LOCATION &&
            d != DEFAULT_BORDER_DISTANCE) {
            dst[tid] = (l == 1 && d == INFINITY) ? -1 : (d > 0.0 ||
d < 0.0) ? d : DEFAULT_INTERIOR_DISTANCE;
        }
    }
}

/*
 * Set up for the parallel FSM implementation
 * Determines the total number of levels and

```

Jun 05, 15 10:42

phi3D_cuda.c

Page 4/8

```

* the nodes on each level that are executed
* in parallel.
* Then determines the direction of the sweep,
* chooses an offset, to translate the coordinates
* for sweeping from different directions.
* Finally, calls the CUDA kernel that calculates
* distance field for a 3D grid using Gauss-Seidal
* iterations.
* (1) i = 1:I, j = 1:J, k = 1:K
* (2) i = I:1, j = 1:J, k = K:1
* (3) i = I:1, j = 1:J, k = 1:K
* (4) i = 1:I, j = 1:J, k = K:1
* (5) i = I:1, j = J:1, k = K:1
* (6) i = 1:I, j = J:1, k = 1:K
* (7) i = 1:I, j = J:1, k = K:1
* (8) i = I:1, j = J:1, k = 1:K
*
* This is the sweeping step discussed in
* section 2.3 of the report.
* Arguments:
*   Phi* [in]      - pointer to the phi function
*   int [in]       - number of sweep iterations
*   cudaPitchedPtr [in/out] - pointer to distance array in
*                       device memory
* Returns:
*/
static void fast_sweep(Phi* p, int noOfTimesToSweep, cudaPitchedPtr dst_dPitchPt
r) {

    // Information regarding sweeping and linear indexing
    int meshDim = 3;
    SweepInfo sw;
    sw.xDim = p->x; sw.dx = p->dx;
    sw.yDim = p->y; sw.dy = p->dy;
    sw.zDim = p->z; sw.dz = p->dz;

    int totalLevels = sw.xDim + sw.yDim + sw.zDim;

    // loop till the number of times to sweep
    int fastSweepLoopCount = 1;
    while( fastSweepLoopCount <= noOfTimesToSweep){
        printf("Please wait. Sweeping...[%d/%d]\n", fastSweepLoopCount, noOfTimes
ToSweep);

        for(int swCount = 1; swCount <= 8; ++swCount){
            int start = (swCount == 2 || swCount == 5 || swCount ==
7 || swCount == 8) ? totalLevels : meshDim;
            int end = ( start == meshDim ) ? totalLevels + 1 : meshD
im-1;

            int incr = ( start == meshDim ) ? true : false;

            // sweep offset is used for translating the 3D coordinat
es

            // to perform sweeps from different directions
            sw.xSweepOff = (swCount == 4 || swCount == 8) ? sw.xDim
+ 1 : 0;
            sw.ySweepOff = (swCount == 2 || swCount == 6) ? sw.yDim
+ 1 : 0;
            sw.zSweepOff = (swCount == 3 || swCount == 7) ? sw.zDim
+ 1 : 0;

            for(int level = start; level != end; level = (incr) ? le
vel+1 : level-1){
                int xs = max(1, level-(sw.yDim + sw.zDim)), ys =
max(1, level-(sw.xDim + sw.zDim));
                int xe = min(sw.xDim, level-(meshDim-1)), ye =
min(sw.yDim, level-(meshDim-1));

                int xr = xe-xs + 1, yr = ye-ys + 1;
                int tth = xr * yr; // Total number of threads ne
eded

```


Jun 05, 15 10:42

phi3D_cuda.c

Page 5/8

```

        dim3 bs(16, 16, 1);
        if(tth < 256){
            bs.x = xr;
            bs.y = yr;
        }
        dim3 gs(iDivUp(xr, bs.x), iDivUp(yr, bs.y), 1);

        sw.level = level;
        sw.xOffset = xs;
        sw.yOffset = ys;

        fast_sweep_kernel<<<gs, bs>>>(dst_dPitchPtr, sw)

        cudaThreadSynchronize();
        cudaCheckError();

    }
    printf("Sweeping finished!.....[%d/%d]\n", fastSweepLoopCount, noOfTimesT
oSweep);
    ++fastSweepLoopCount;
}

/*
 * Kernel for fast sweeping method
 * Arguments:
 *   cudaPitchedPtr [in/out] - pointer to distance array in
 *                           device memory
 *   SweepInfo [in] - sweep information
 */
__global__ void fast_sweep_kernel(cudaPitchedPtr dst_dPitchPtr, SweepInfo s) {
    int x = (blockIdx.x * blockDim.x + threadIdx.x) + s.xOffset;
    int y = (blockIdx.y * blockDim.y + threadIdx.y) + s.yOffset;
    if(x <= s.xDim && y <= s.yDim) {
        int z = s.level - (x+y);
        if(z > 0 && z <= s.zDim){
            int i = abs(z-s.zSweepOff);
            int j = abs(y-s.ySweepOff);
            int k = abs(x-s.xSweepOff);

            char *devPtr = (char *) dst_dPitchPtr.ptr;
            size_t pitch = dst_dPitchPtr.pitch;
            size_t slicePitch = pitch * (s.yDim + 2);

            double *c_row = (double *) ( (devPtr + i * slicePitch) +
j * pitch);
            // center row
            double center = c_row[k];
            // center distance
            double left = c_row[k-1];
            // left distance
            double right = c_row[k+1];
            // right distance
            double up = ((double *) ( (devPtr + i * slicePitch) + (j
-1) * pitch) )[k];
            // upper distance
            double down = ((double *) ( (devPtr + i * slicePitch) +
(j+1) * pitch) )[k];
            // lower distance
            double front = ((double *) ( (devPtr + (i-1) * slicePitc
h) + j * pitch) )[k];
            // front distance
            double back = ((double *) ( (devPtr + (i+1) * slicePitch
) + j * pitch) )[k];
            // back distance

            double minX = min(left, right);
            double minY = min(up, down);
            double minZ = min(front, back);
            c_row[k] = solve_eikonal(center, minX, minY, minZ, s.dx,
s.dy, s.dz);
        }
    }
}

```

Friday June 05, 2015

phi3D_cuda.c

Jun 05, 15 10:42

phi3D_cuda.c

Page 6/8

```

}

/*
 * Solves the Eikonal equation at each point of the grid.
 * Arguments:
 *   double - current distance value
 *   double - minimum distance in the x-direction
 *   double - minimum distance in the y-direction
 *   double - minimum distance in the z-direction
 *   double - spacing in the x-direction
 *   double - spacing in the y-direction
 *   double - spacing in the z-direction
 */
__device__ double solve_eikonal(double cur_dist, double minX, double minY, doubl
e minZ, double dx, double dy, double dz) {
    double dist_new = 0;
    double m[] = { minX, minY, minZ };
    double d[] = { dx, dy, dz };

    // sort the mins
    for(int i = 1; i < 3; i++){
        for(int j = 0; j < 3-i; j++) {
            if(m[j] > m[j+1]) {
                double tmp_m = m[j];
                double tmp_d = d[j];
                m[j] = m[j+1]; d[j] = d[j+1];
                m[j+1] = tmp_m; d[j+1] = tmp_d;
            }
        }
    }

    // simplifying the variables
    double m_0 = m[0], m_1 = m[1], m_2 = m[2];
    double d_0 = d[0], d_1 = d[1], d_2 = d[2];
    double m2_0 = m_0 * m_0, m2_1 = m_1 * m_1, m2_2 = m_2 * m_2;
    double d2_0 = d_0 * d_0, d2_1 = d_1 * d_1, d2_2 = d_2 * d_2;

    dist_new = m_0 + d_0;
    if(dist_new > m_1) {

        double s = sqrt(- m2_0 + 2 * m_0 * m_1 - m2_1 + d2_0 + d2_1);
        dist_new = ( m_1 * d2_0 + m_0 * d2_1 + d_0 * d_1 * s ) / (d2_0 + d2_1);

        if(dist_new > m_2) {

            double a = sqrt(- m2_0 * d2_1 - m2_0 * d2_2 + 2 * m_0 * m_1 * d2_2
- m2_1 * d2_0 - m2_1 * d2_2 + 2 * m_0 * m_2 * d2_1
- m2_2 * d2_0 - m2_2 * d2_1 + 2 * m_1 * m_2 * d2_0
+ d2_0 * d2_1 + d2_0 * d2_2 + d2_1 * d2_2);

            dist_new = (m2_2 * d2_0 * d2_1 + m_1 * d2_0 * d2_2 + m_0 * d2_1 * d2_
2 + d_0 * d_1 * d_2 * a) /
                (d2_0 * d2_1 + d2_0 * d2_2 + d2_1 * d2_2);
        }
    }

    return min(cur_dist, dist_new);
}

/*
 * Sets up the kernel call to set the inside distance
 * values in the array to negative based on location.
 * Arguments:
 *   Phi3D_d* [in/out] - pointer to phi function (GPU)
 *   Grid3D [in] - dimensions of grid
 * Returns:
 */
static void set_distance_negative_inside(Phi3D_d *d_p, Grid3D g3d) {

```

3/4

Jun 05, 15 10:42

phi3D_cuda.c

Page 7/8

```

    // Setup 3D-Grid and 3D-Block for Kernel launch
    // Running 256 threads per block
    dim3 bs(8, 8, 8);
    dim3 gs(iDivUp(g3d.x, bs.x), iDivUp(g3d.y, bs.y), iDivUp(g3d.z, bs.z));
    set_distance_negative_inside_kernel<<gs, bs>>>(d_p->dst_dPitchPtr, d_p->loc_dPitchPtr, g3d);
    cudaThreadSynchronize(); cudaCheckError();
}

/*
 * Kernel for fast sweeping method
 * Arguments:
 *   cudaPitchedPtr [out] - pointer to distance array in
 *                         device memory
 *   cudaPitchedPtr [in]  - pointer to location array in
 *                         device memory
 *   Grid3D [in]         - dimensions of grid
 */
__global__ void set_distance_negative_inside_kernel(cudaPitchedPtr dst_dPitchPtr,
    cudaPitchedPtr loc_dPitchPtr, Grid3D g3d) {
    int x = blockIdx.x*blockDim.x+threadIdx.x + 1;
    int y = blockIdx.y*blockDim.y+threadIdx.y + 1;
    int z = blockIdx.z*blockDim.z+threadIdx.z + 1;

    if (x < g3d.x-1 && y < g3d.y-1 && z < g3d.z-1 ) {

        char *dist_devPtr = (char *) dst_dPitchPtr.ptr;
        size_t dist_pitch = dst_dPitchPtr.pitch;
        size_t dist_slicePitch = dist_pitch * g3d.y;
        char* dist_slice = dist_devPtr + z * dist_slicePitch;
        double* dist_row = (double *) (dist_slice + y * dist_pitch);

        char *loc_devPtr = (char *) loc_dPitchPtr.ptr;
        size_t loc_pitch = loc_dPitchPtr.pitch;
        size_t loc_slicePitch = loc_pitch * g3d.y;
        char* loc_slice = loc_devPtr + z * loc_slicePitch;
        int* loc_row = (int *) (loc_slice + y * loc_pitch);

        if(loc_row[x] == 1) dist_row[x] = -1;
    }
}

/*
 * Copies location and distance array memory
 * from device pinned memory to device memory
 * Arguments:
 *   Phi3D_d* [in/out] - pointer to d_Phi (Device - Destination)
 *   double* [in]      - pointer to distance (Pinned - Source)
 *   int* [in]         - pointer to location (Pinned - Source)
 * Returns:
 */
static void cudaMemcpy3D_P2D(Phi3D_d *d_p, double *dst, int *loc) {
    cudaMemcpy3DParms mcp = { 0 };
    mcp.kind = cudaMemcpyDeviceToDevice;

    // copy parameters for distance
    mcp.dstPtr = d_p->dst_dPitchPtr;
    mcp.srcPtr.ptr = dst;
    mcp.srcPtr.pitch = d_p->dst_ext.width;
    mcp.srcPtr.xsize = (size_t) (d_p->dst_ext.width/sizeof(double));
    mcp.srcPtr.ysize = d_p->dst_ext.height;
    mcp.extent = d_p->dst_ext;
    cudaMemcpy3D(&mcp); cudaCheckError();

    // copy parameters for location

```

Jun 05, 15 10:42

phi3D_cuda.c

Page 8/8

```

    mcp.dstPtr = d_p->loc_dPitchPtr;
    mcp.srcPtr.ptr = loc;
    mcp.srcPtr.pitch = d_p->loc_ext.width;
    mcp.srcPtr.xsize = (size_t) (d_p->loc_ext.width/sizeof(int));
    mcp.srcPtr.ysize = d_p->loc_ext.height;
    mcp.extent = d_p->loc_ext;
    cudaMemcpy3D(&mcp); cudaCheckError();
}

/*
 * Copies location and distance array memory
 * from device memory to pinned memory
 * Arguments:
 *   Phi3D_d* [in/out] - pointer to d_Phi (Device - Source)
 *   double* [in]      - pointer to distance (Pinned - Destination)
 *   int* [in]         - pointer to location (Pinned - Destination)
 * Returns:
 */
static void cudaMemcpy3D_D2P(Phi3D_d *d_p, double *dst, int *loc) {
    cudaMemcpy3DParms mcp = { 0 };
    mcp.kind = cudaMemcpyDeviceToDevice;

    // copy parameters for distance
    mcp.srcPtr = d_p->dst_dPitchPtr;
    mcp.dstPtr.ptr = dst;
    mcp.dstPtr.pitch = d_p->dst_ext.width;
    mcp.dstPtr.xsize = (size_t) (d_p->dst_ext.width/sizeof(double));
    mcp.dstPtr.ysize = d_p->dst_ext.height;
    mcp.extent = d_p->dst_ext;
    cudaMemcpy3D(&mcp); cudaCheckError();

    // copy parameters for location
    mcp.srcPtr = d_p->loc_dPitchPtr;
    mcp.dstPtr.ptr = loc;
    mcp.dstPtr.pitch = d_p->loc_ext.width;
    mcp.dstPtr.xsize = (size_t) (d_p->loc_ext.width/sizeof(int));
    mcp.dstPtr.ysize = d_p->loc_ext.height;
    mcp.extent = d_p->loc_ext;
    cudaMemcpy3D(&mcp); cudaCheckError();
}

```