A Fast Iterative Method for Solving the Eikonal Equation on Triangulated Surfaces

Zhisong Fu, Won-Ki Jeong, Yongsheng Pan, Robert M. Kirby, Ross T. Whitaker

Motivation

In this project, we consider the numerical solution of the Eikonal equations, a special case of nonlinear Hamilton-Jacobi partial differential equations (PDEs), defined on a three dimensional surface with a scalar speed function:

$$H(\mathbf{x}, \Delta\phi) = |\Delta\phi(\mathbf{x})|^2 - \frac{1}{f^2(\mathbf{x})} = 0 \quad \forall \mathbf{x} \in S \subset \Omega$$

S is a surface domain. The solution of this equation simulates travel time of the wave propagation with speed f at x from some source points whose values are zero. The Eikonal equation appears in various Applications, such as computer vision, image processing, computer graphics, geoscience, and medical image analysis.

Background

1.Mesh Fast Iterative Method(meshFIM) [1]

◆An iterative computational technique to solve the Eikonal equation efficiently on parallel architectures.

◆This method relies on a modification of a label-correcting method.

◆The core elements for our FIM based method are:

(1) Upwind scheme: calculate the value at a vertex with the values of the solved vertices.

(2) Active list management: Active list contains the patches which has wave front vertices. If a active patch is convergent, it is removed from the Active list and its neighbor patches are added to this list.

(3) Patch-based iteration: divide the whole mesh into patches to fit into GPU cores.

(4) Triangle-based Jacobi update: update all the triangles inside a patch concurrently with parallel threads and each thread updates values of the three triangle vertices.

2.Method description

(1) Firstly, partition the mesh into patches.

(2) Add the patches which contain the source vertices to active list.

(3) Assign each patch to a GPU stream processor and iterate multiple times for each patch.

(4) Then check if a patch is convergent which means all the vertices of this patch are convergent. Remove a convergent patch from the active list and add its neighbor patches.

(5) Check if the patches in active list are already convergent, if so remove.

(6) Iterate again.

3.Suitability for GPU

◆Each vertex updates independently

◆According to the algorithm, update operation can be completed concurrently

◆Computing only depends on the neighbors of same facet at every time step

Implementation

1.Partition

◆In the process of partitioning, we will use edges instead of coordinates, thus our partition can be viewed as the graph-based partition

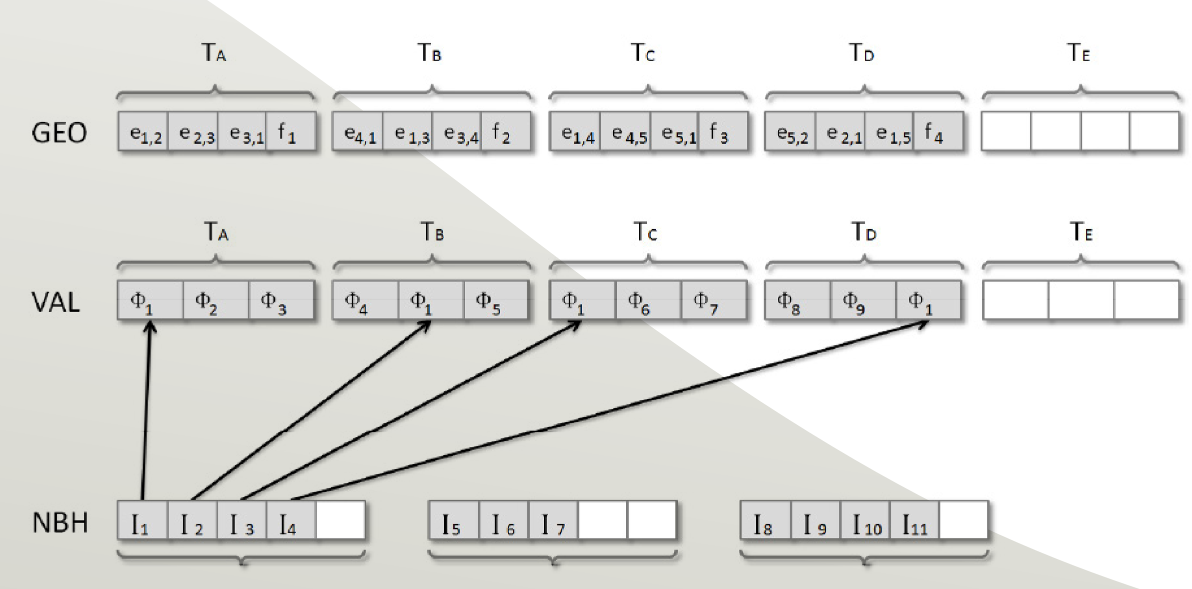
◆We use METIS [2] as partition tool (See the figure below for a partition result of a dragon)

2.Triangle-based data structure

◆GEO: divided into sub segment for each patch and each patch subsegment contains geometric data and speed information for each triangle: three floats for edge lengths of the triangle and one float for speed.

◆VAL: hold all the vertex values(float) of all triangles patch by patch.

◆NBH: an integer array with each integer element representing an index of a vertex value in the value array.

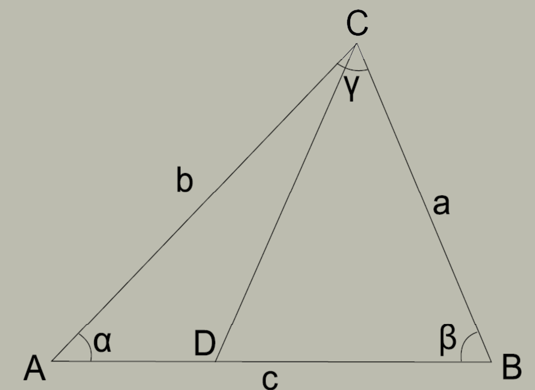


Data structure figure

◆GEO is in global memory and VAL and NBH are copied into shared memory for multiple updates.

3.Localsolver

As in the figure below, local solver calculate the value of a vertex of the triangle ΔABC from the other two vertices. Without loss of generality we only talk about calculating value of C, T_C, from values of A and B, T_A, T_B . f is the speed.



Assume wave propagation direction is from D to C, and $T_D = \lambda(T_{AB}) + T_A$ we get:

$$\begin{aligned} T_C &= T_D + T_{DC} \\ &= \lambda(T_{AB}) + T_A + f\|DC\| \\ &= \lambda(T_{AB}) + T_A + f\|\vec{AC} - \lambda\vec{AB}\| \end{aligned}$$

And the location of D must minimize T_C , so let: $\frac{dT_C(\lambda)}{d\lambda} = 0$,We can solve for λ and then substitute into above equation to get T_C .

Algorithm

Algorithm 3.2: PATCHFIM($\mathbf{VAL}_{in}.\mathbf{VAL}_{out}, L, P$)

```
comment: L: active list of patches, P: set of all patches
while L is not empty
    do { MainUpdate(L, C_v, VAL_in, VAL_out)
       { CheckNeighbor(L, C_v, C, VAL_in, VAL_out)
       { UpdateActiveList(L, P, C)
```

Algorithm 3.3: MAINUPDATE($L, C_v, \mathbf{VAL}_{in}, \mathbf{VAL}_{out}$)

```
comment: 1. Main iteration
for each p ∈ L in parallel
    do { for i = 1 to n
       {     do { for each t ∈ p in parallel
       {        {     do { VAL_out(t) ← LocalSolver(VAL_in(t))
       {        {        { reconcile solutions in t
       {        { update C_v(p)
       {        { swap VAL_in(t) and VAL_out(t)
       {        { reconcile solutions in p
```

Algorithm 3.4: CHECKNEIGHBOR($L, C_v, C, \mathbf{VAL}_{in}, \mathbf{VAL}_{out}$)

```
comment: 2. Check neighbors
for each p ∈ L in parallel
    do {C(p) ← reduction(C_v(p))

for each p ∈ L in parallel
    do { if C(p) = TRUE
       {    then { for each adjacent neighbor of p_nb of p
       {         {     do {add p_nb to L

for each p ∈ L in parallel
    do { for each t ∈ p in parallel
       {     do { VAL_out(t) ← LocalSolver(VAL_in(t))
       {        { reconcile solutions in t
       { update C_v(p)
       { swap VAL_in(t) and VAL_out(t)
       { reconcile solutions in p

for each p ∈ L in parallel
    do {C(p) ← reduction(C_v(p))
```

Algorithm 3.5: UPDATEACTIVELIST(L, P, C)

```
comment: 3. Update active list
clear(L)
for each p ∈ P
    do { if C(p) = FALSE
       {    then insert p to L
```

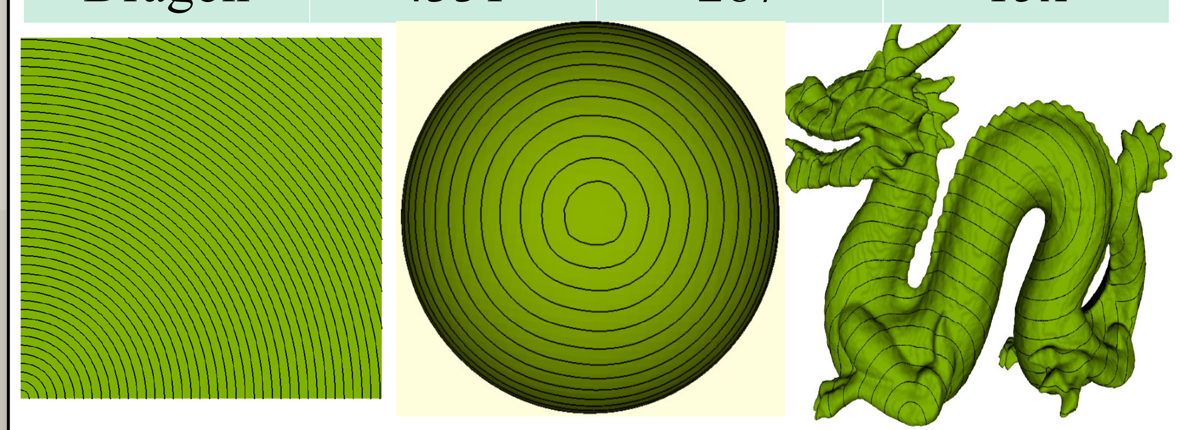
Result

◆CPU: Intel i7 920, 2.66GHz, 8M cache

◆GPU: Nvidia GTX 275, 1.404GHz, 240 core

We test running time(ms) for a CPU version of meshFIM to compare with GPU version on three different meshes:

Mesh	CPU	GPU	Speedup
Square	6562	201	33x
Sphere	8591	415	21x
Dragon	4331	287	15x



References

1. A Fast Iterative Method For Eikonal Equations. Won-Ki Jeong, Ross Whitaker.

2. METIS: A Family of Multilevel Partitioning Algorithms. http://glaros.dtc.umn.edu/gkhome/views/metis

SCI INSTITUTE • EXHIBIT • EXPLORE • EXCITE • EXPERIENCE • EXCHANGE

THE UNIVERSITY OF UTAH

www.sci.utah.edu