Solving EikonalEquations on Triangulated Surface Mesh with CUDA

Zhisong Fu, Ross T. Whitaker
School of Computing, University of Utah

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.Fast iteraive method(FIM)[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

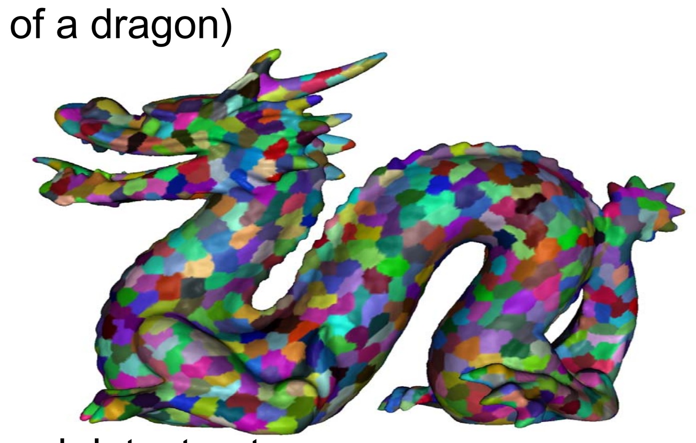
◆According to the algorithm, update operation can be completed concurrently

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)

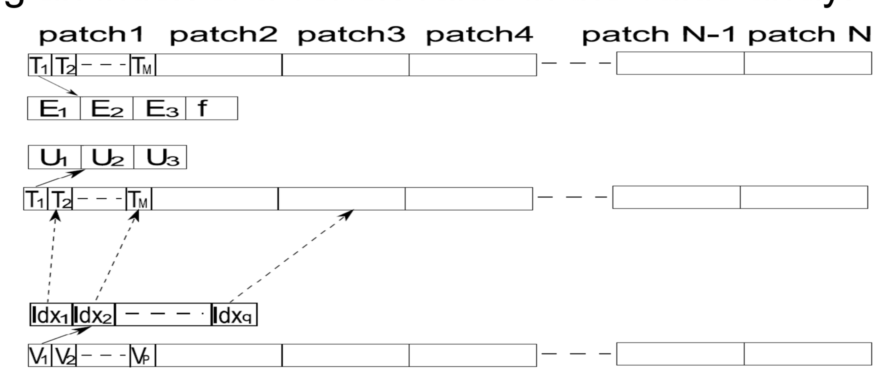


2.triangle-based datastructure

◆Geo-array: divided into subsegment 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.

◆Input-value-array and Output-value-array: hold all the vertex values(float) of all triangles patch by patch.

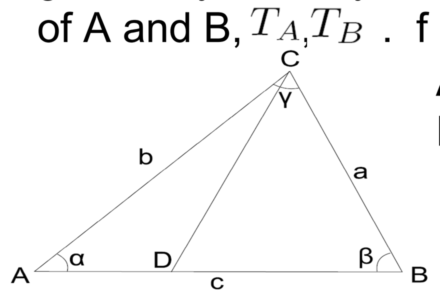
◆index-array: an integer array with each integer element representing an index of a vertex value in the value array.



◆Geo-array and index-array are in global memory and Value-arrays are copied into shared memory for multiple updates.

3.Local solver

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

Algorithm

Algorithm 2 patchFIM(Geo-array, Input-value-array, Output-value-array, L, P)(L: active list of patch, P: set of all patches)

```
while L is not empty do
  comment: main iteration
  for all p ∈ L in parallel do
    for 1 to a certain user defined number do
      for all t ∈ p in parallel do
        Output − value − array[t] ← LocalSolver(Input − value − array[t])
        reconcile the values
      end for
      update C_n(p)
      reconcile the values
    end for
  end for
  comment: reduction
  for all p ∈ L in parallel do
    C(p) ← recuction(C_n(p))
  end for
  comment: Check neighbors
  for all p ∈ L in parallel do
    if C(p) = true then
      for all adjacent neighbor of p_nb of p do
        add p_nb to L
      end for
    end if
  end for
  for all p ∈ L in parallel do
    for all t ∈ p in parallel do
      Output − value − array[t] ← LocalSolver(Input − value − array[t])
      reconcile the values
    end for
    update C_n(p)
    reconcile the values
  end for
  comment: reduction again
  for all p ∈ L in parallel do
    C(p) ← recuction(C_n(p))
  end for
  comment: update active list
  clear(L)
  for all p ∈ P do
    if C(p) = FALSE then
      insert p to L
    end if
  end for
end while
```

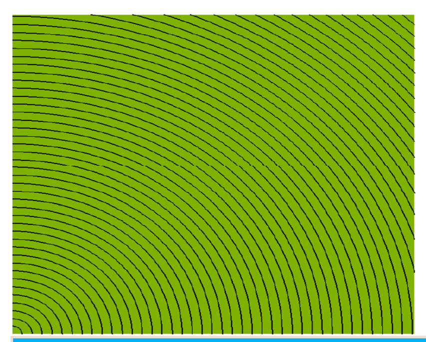
Result

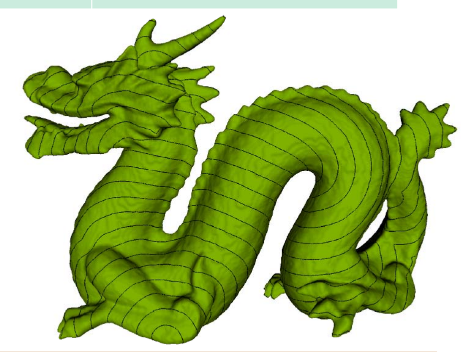
◆CPU: Intel i7 965, 3.2GHz, 8M cache

◆GPU: Nvidia GeForce GTX 275, 1.4GHz

We test running time(ms) for a CPU version of FIM to compare with GPU version on three different meshes(See figures below for results):

Mesh	CPU	GPU	Speedup
Flat square	13409	301	44.55
Bumpysphere	1151	51	22.57
dragon	10143	331	30.64





Reference

1. A Fast Iterative Method For EikonalEquations. Won-KiJEONG, Ross Whitaker.

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