ISSN:0975-9646

Parallelizing 1D Midpoint Displacement Algorithm for Terrain Generation

Yadav Mayank Subhash#1, Prakash Tripathi#2, P.S.Rao#3

#Department of Applied Mathematics, Indian School of Mines, Dhanbad, India

***Abstract*— 1D Midpoint Displacement Algorithm is one of the most fundamental algorithms used for terrain generation in the field of Computer Graphics. With the advent of Parallel Computing and GPUs it is now possible to use the high performance of multiple processors to increase the speed of computation. We are proposing a parallel implementation of 1D Midpoint Displacement Algorithm for Terrain Generation in this paper. The basic idea is to solve all the sub problems in the sequential algorithm in parallel using GPUs.**

***Keywords*— parallel, algorithm, terrain generation, GPU computing, 2D midpoint displacement algorithm.**

I. Introduction

1D Midpoint Displacement algorithm is one of the most fundamental algorithms used for terrain generation in the field of Computer Graphics. It is used widely by programmers around the world because of its simplicity, efficiency and the quality of result it produces. This algorithm is recursively called till a result with desired resolution or smoothness is not obtained. Now, this takes high computation time when the smoothness expected is of high quality. In order to tackle this problem, we need to use the power of GPUs (Graphics Processing Units) that can help in solving the sub problems in parallel resulting in smaller computation time. We are going to describe the sequential algorithm and then proceed towards its parallel implementation.

II. Sequential Algorithm

The 1D midpoint displacement algorithm [1] starts with a line between two points. These two points are considered to define the horizontal axis of the 2D final geometry. The midpoint of the two initial points is displaced vertically be a random amount. The two new line segments are then divided in half and the midpoints of these segments are displaced by a random amount. This process is repeated until the desired level of detail is attained with the range of the random displacement being reduced in each pass. For example, if the random range was reduced by half for each pass and the original range of the random displacement was from -1.0 to 1.0, then the range for the second pass with two midpoints would be from -0.5 to 0.5 and -0.25 to 0.25 for the third pass with four midpoints. The amount by which the range of random numbers is reduced in each pass is controlled. In fact the amount by which the range decreases is one of the factors which controls the final geometry.

The figure 1 below shows the three passes in midpoint displacement algorithm.

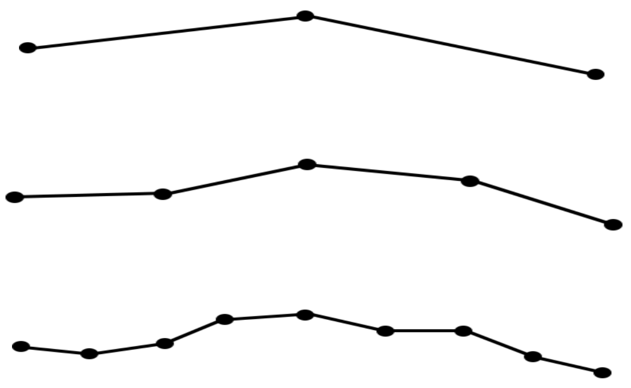

Fig. 1 Example of 1D midpoint displacement algorithm

The brief overview of how midpoint displacement algorithm works is given below:

```
For Each Segment
    Divide in half and average the middle point
      Add a random value to the mid-point
      Reduce the random range
Repeat until segments are too small to continue.
```

The computation time for the sequential implementation can be very high when high level of smoothness is needed. In this age of high-end hardware, it is not favourable to spend so much computation time to generate a terrain. It can be observed that the process for determination of value at middle point is always the same for any point. We will use this property and process all the intermediate segments (sub problems) in parallel which will drastically reduce the time taken to generate the terrain.

We will now give the overview of NVidia GPUs and then propose our parallel algorithm for 1D Midpoint Displacement.

III. NVidia GPUs And Cuda

A Graphics Processing Unit is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the processing speed for a task being executed on it. Modern GPUs are very efficient at manipulating computer graphics and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where the processing of large blocks of data is done in parallel. NVidia has made a lot of progress in the creation of GPUs and their API named CUDA is widely used for performing high end parallel computation on NVidia GPUs. The CUDA API is very well documented and easy to use. The user only needs to adhere to the syntax and the API will manage the parallel computation on its own.

A function definition preceded by the keyword __global__ declares kernel, which is called on host and executed on the GPU on each thread. We can divide the whole problem space in blocks and grids as per our convenience and CUDA runs the operation on these in parallel. This accelerates the process rapidly and a lot of processing time is saved.

In our application we will do computation in single block with number of threads equal to number of points to be obtained for terrain generation. This will be sufficient in reducing the time needed to generate terrain using 1D midpoint displacement algorithm.

IV. Parallel 1D Midpoint Displacement Algorithm

In the parallel algorithm, we will harness the power of NVidia GPUs by using CUDA API. We will assume the number of points on which the value is to be calculated to be of the form $2^N + 1$. We will initialize the values for all the elements at 0, and then we will start the process of calculating the values at each points in subsequent passes.

The essential parts of our parallel algorithm are:

1) Locking mechanism: We are keeping a global integer array named lockVal[ ] of the size equal to the number of points to be calculated. Initially, all the values in the lockVal[ ] is set to 0. When calling the function, we will set isLocked[N/2] to 1. We will also initialize the values at two extreme points to some constant (say 0).

2) Computation: When access to operation at a thread (say threadIdx) is unlocked i.e. when lockVal[threadIdx] > 0, the value at the corresponding thread is calculated by using midpoint displacement method. Then, the thread unlocks the access for operation at threads (threadIdx – threadIdx/2, threadIdx + threadIdx/2) by setting their lockVal[ ] value to (lockVal[threadIdx]+1).

Using these two operations, we can complete our terrain generation using parallel 1D midpoint displacement algorithm using CUDA API.

We will use global integer array lockVal[] which will be initialized to 0. The operation of midpoint displacement algorithm will be performed only when the value at the corresponding thread is greater than 0. The kernel code for midpoint displacement algorithm is given alongside.

```
__global__ void midpoint_disp_parallel (double*
terrain_array, double displacement, double roughness) {
  int flag = true;
    while(flag) {
      if(lockVal[threadIdx] > 0) {
        int diff = N/pow(2,lockVal[threadIdx]);
        double rand_num = (rand()*10000)/10000.0;
        double    change    =    (rand_num*2.0    -
1)*displacement;
        int left = threadIdx – diff;
        int right = threadIdx + diff;
        terrain_array[threadIdx] = (terrain_array[left] +
terrain_array[right])/2.0 + change;
        displacement = displacement*roughness;

        flag = false;
        lockVal[threadIdx + threadIdx/2] =
lockVal[threadIdx] + 1;
        lockVal[threadIdx – threadIdx/2] =
lockVal[threadIdx] + 1;
      }
    }
  }
```

We will call this function with one block and number of threads equals to total elements in the array named *terrain_array*. The function will be called using two parameters namely *displacement* and *roughness*. Depending on these two values different terrains will be generated. The values obtained in the *terrain_array* can be fed in terrain rendering function which will display the 2D terrain obtained from the algorithm above.

V. Result

On feeding the obtained values from the above algorithm in one of our terrain rendering function with *displacement* value set at 50.0 and *roughness* 0.55 we obtained the following output as in Figure 2:

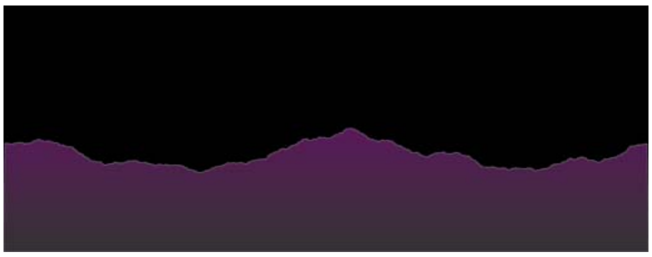

Fig. 2 Output for displacement = 50 and roughness = 0.55.

The algorithm takes time of O(lg(N)), where N is the size of *terrain_array*. This is a very big improvement in comparison to the sequential algorithm which has linear time complexity.

VI. Conclusion

The 1D midpoint displacement algorithm has been successfully implemented in parallel and the time consumed for terrain generation has been reduced drastically by using NVidia GPU and CUDA API. Different terrains can be generated by changing the value of *displacement* and *roughness* parameter in the algorithm. For generating terrains that are more detailed, we must use the *terrain_array* of larger size. Also, the best performance of this algorithm will be when the size of *terrain_array* is of the form $2^N + 1$ because there won’t be any kind of memory collision.

VII. Future Work

The parallel 1D midpoint displacement algorithm for terrain generation can be extended to 2D and 3D midpoint displacement algorithm as well.

Acknowledgment

We are thankful for the guidance and support of our mentor Dr. P.S. Rao, Assistant Professor, Department of Applied Mathematics.

References

[1] J. Jilesen et al., *Three-dimensional midpoint displacement algorithm for the generation of fractal porous media,* Journal Computers & Geosciences. Tarrytown, NY, US: September 2012, vol. 46.

[2] Jason Sanders and Edward Kandrot, "CUDA by Example", *Addison-Wesley, 2012.*