Geraldo Braho
North American University, Department of Computer Science
Houston, TX, USA
gbraho@na.edu

A Study on GPU Terrain Generation using Brute Force and The Enhancement of The Acclaimed Bubble Sort Algorithm

Abstract—This work presents a summary of two peer reviewed papers that fall into dissimilar areas of Algorithmic Study, Implementation, and Time and Space Efficiency. These two topics cover (1) using a Brute Force algorithmic approach for Video Game Terrain Generation efficiency with respect to CPU overhead and GPU rendering capabilities and (2) Enhancing a Bubble Sort Algorithm by introducing a new sorting method called RTBS (Run Time Bubble Sort).
Keywords — Algorithms, Efficiency, Brute Force, Sorting, Bubble Sort.

Introduction

Have we ever wondered how the terrain of video games or 3D designing programs are generated, or how their processing power affects our devices? Their loading times? Real time rendering? Whether we are viewing in high or low quality? The use of Brute Force spans a vaster spectrum of computing problems than we might think we know. Sorting is another significant aspect of algorithmic study. Sorting technology can be found in almost every computing device from cell phones, PDA's, game consoles, webpages, search engines, to servers, text editors, and a variety of a lot more devices and systems. We should look into two summarized peer reviewed papers of both algorithmic topics, Brute Force and Bubble Sorting.
Aspects Learnt

- *1)* Brute force has some impressive advantages: *Although it may seem like one algorithmic approach out of a couple that can be used to solve the instance of a problem but rather we have come to understand that it is more significant than that. Although it may be seen as the least preferred method to use, in some cases, it is a better approach to use. Terrain Generation from the processing power of a GPU can be made a lot more efficient by using Brute Force capabilities within the Game Designing Engine that the graphical generation was design with. We have come to learn that it is not always a disadvantage to use Brute force when it comes to problem solving.*
- *2)* Internal Algorithms vs External Algorithms in reference to memory usage: *Another aspect that was gleaned is in regards to sorting and memory usage. Sorting algorithms can either be internal or external. External ones require a lot more memory while Internal ones require the opposite. The reason being External sorting algorithms make use a large amount of External memory, Internal sorting algorithms ply the original array which only needs a few bytes for transitory variables and stack. Surprisingly we also learnt that all basic sorting algorithms function internally with the exception of merge sort.*

SUMMARY

- *1)* Terrain Generation Bruteforce Algorithm: *Various algorithms increase huge terrain rendering from quad tree based frustum culling to continuous Level of Details (LOD). The most successful techniques are said to be combinations of two or more others. The main purpose of these rendering processes is to reduce the amount of triangle being passed to the graphic pipeline. A bunch of techniques reduce this by surveying triangles that are not visible behind other triangles. Modern graphic processors aid with their capability of surveying millions of triangles per second and also carry other task such as audio sampling at the same time. All this data is processed in their raw formats before reaching the display or output device responsible for projection. Accepting and pushing all this raw data could mean the projection of redundant and actually unused data can be provided within the terrain. Sub algorithms must come into play to fulfill the individual purpose of all these projections and processes. Much of the complex algorithms become outmoded due to their overhead of the main processor. This technique of rendering does not require per triangle testing to decide it is visible or not so the sob is offered to the GPU to handle. This method of rendering is called Brute-Force with the purpose being "just send it to the GPU and it will handle it). With the use of Brute-Force, comes an optimization technique called Quadtree Rendering which has simplest procession time and is time efficient.*

Terrain can be representing in various ways but the simplest is in the form of a regular grid with differential heights (Fig. 1).

Fig.1: The layout of the terrain from top. Based on quads and each quad consists of two triangles. The empty circles show indexing method.

Each circle in the figure represents one vertex in 3D space. (each has a x,y,z component). The amount of vertices in either dimension is set to be odd to make the number of quads even. Distance between the neighboring vertices is equivalent to each other thus repressing a two dimensional array of heights. With Quad Tree View Frustum Culling is relatively fast and does not provoke CPU overhead. The concept is based on the view frustum (the field of view) which can be camera (Fig.2).

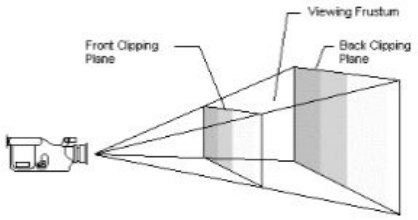


Fig.2 View frustum example. The viewable area is between both planes (taken from Microsoft® DirectX® documentation)

Fig.2 View frustum example. The viewable area is between both planes (taken from Microsoft® DirectX® documentation)

The purpose of the Quad tree technique is that every single terrain patch can be represented by four little pieces. (Fig. 3).

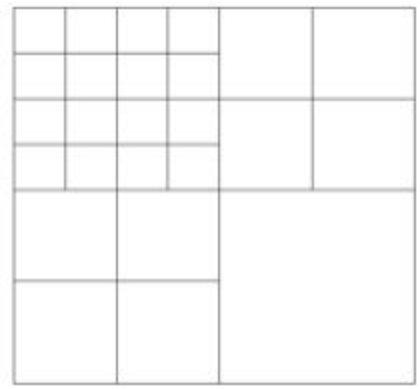
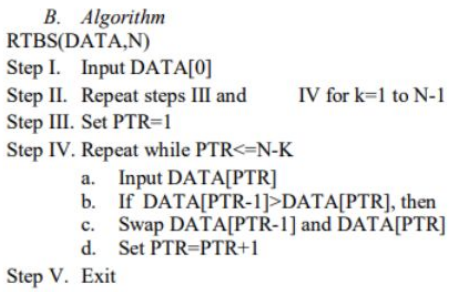


Fig. 3: Quad-tree – graphical representation. Each patch is made of four smaller pieces and so on…

Thus not all patches should be examined in a linear manner but the tree structure could be used. Let's take an assumption that after a collision test we determine that one big patch is not fully, even partially in the viewing frustum so we can stop examining the lower leaves of this tree (the patches four patches that build this big one). This leads to cutting down the number of tests to minimum and gives a very high performance. The only problem is that it's not too accurate and does not minimize the number of primitives drawn to the complete minimum but as we discussed before the new graphic processors take the responsibility of rendering more triangles.

2) Run Time Bubble Sort: The concept of the RTBS (Run Time Bubble Sort) Algorithm Technique is as follows: The sorting starts when the user gives the first two elements. The first two elements are sorted before the user gives the third element. The initial three elements are sorted before the user gives the fourth element and so on the process continues until N number of elements to be sorted. With this enhancement to the bubble sort we are decreasing the traversal time of the elements from the beginning. It is a comparison sort with the property of stability.
In Fig. 4. Is the pseudocode of the algorithm.

Fig.4: Pseudocode demonstrating the Run Time Bubble Sort individual sorting steps with comparison based on user input order.



B. Algorithm

RTBS(DATA,N)
Step I. Input DATA[0]
Step II. Repeat steps III and IV for k=1 to N-1
Step III. Set PTR=1
Step IV. Repeat while PTR<=N-K
a. Input DATA[PTR]
b. If DATA[PTR-1]>DATA[PTR], then
c. Swap DATA[PTR-1] and DATA[PTR]
d. Set PTR=PTR+1
Step V. Exit

ig.4: Pseudocode demonstrating the Run Time Bubble Sort individual sorting steps with comparison based on user input order.

In order to test this proposed algorithm for its efficiency the algorithm was implemented in C language on core i3 machine with operating system window 8. To calculate the execution time of both algorithms, clock() function is used. Both algorithms are compared on the same elements of unordered list. In order to make a comparison of the proposed algorithms with the existing Bubble sort, a number of tests were conducted for small as well as large number of elements. Comparison is shown in tabular form as following:

TABLE I
Elapsed Time for the Proposed RTBS algorithm and the Existing Bubble Sort Algorithm

No. of elements	RTBS (in ms)	Bubble sort (in ms)
10	40	40
100	39	52
1000	36	51
10000	164	191

As shown in Table 1 above, 10 unordered elements have been taken and sort them using proposed RTBS as well as Bubble sort. The elements are selected randomly using random() function. The elapsed time was same for both sorts. As the number of elements increases up to 100, efficiency of proposed algorithm has been shown. The elapsed time using Run Time Bubble sort was less than using existing Bubble sort. The difference of elapsed time grows as number of elements increases.

References

http://web.a.ebscohost.com/ehost/resultsadvanced?vid=9&sid=c11aac4f-198c-4597-b86b-d5f0a986fa6a%40sessionmgr4010&bquery=XX+%22brute%22%5b100%5d+AND+(XX+%22terrain%22%5b79%5d+OR+XX+%22algorithm%22%5b51%5d+OR+XX+%22force%22%5b47%5d+OR+XX+%22generation%22%5b46%5d)&bdata=JmRiPWFjaSZjbGkwPUZUJmNsdjA9WSZjbGkxPVJWJmNsdjE9WSZ0eXBlPTEmc2l0ZT1laG9zdC1saXZlDeLoura M, Game Programming Gems, Charles River Media, Inc., 2000
http://www.ijcttjournal.org/Volume14/number-1/IJCTT-V14P109.pdf