An Improvement of Zalik's Containment Test Algorithm and Applications on Terrain Query

Le Hoang Son, Nguyen Duy Linh and Nguyen Thi Hong Minh

***Abstract*—In this paper, we will herein present an improvement of the containment test algorithm of Zalik et al. (2001) for the purpose of fast computation on the polygon shape datasets over the web environment. Experimental results show that the proposed method is faster than that of Zalik. It will be applied to perform the terrain query on a 3D WebGIS system.**

***Index Terms*—3D WebGIS, containment test, computational geometry, GIS.**

I. Introduction

We begin with some definitions.

Def. 1: A *polygon* U_j is a sequence of two-dimensional points $M_i^j(x_i^j, y_i^j)$, oriented by a specific direction Dt^j, where $i=\overline{1,n_j}$, and n_j is the total number of vertices in the polygon. The sign (-) means the direction of polygon is counterclockwise.

Def. 2: A *polygon shape dataset* can be expressed by a sequence $\{U_j \mid j=\overline{1,l}\}$ where l is total number of polygons, and $U_i \cap U_j = \phi$, $i \neq j$, $i=\overline{1,l}$, $j=\overline{1,l}$.

This article is motivated by looking for a fast containment test algorithm that works with a special geographic dataset namely the polygon shape (Def. 2) over the web environment. It can be used for further advanced analyses on the *Web-based Geographic Information Systems* (WebGIS) or three-dimensional WebGIS (3D WebGIS). The problem is recognized when we process the large polygon shape datasets. A lot of polygons may slow down the checking process and prevent the deployment of any analysis on the WebGIS. A fast containment test algorithm is required in this situation.

There are a number of algorithms dealing with this obstacle such as *Coded coordinate system method* [1], *triangle-based method* [2], *Approximate algorithm* [3], *Grid method* [4], *Convex Decomposition* [5], *Ray crossing* [6], *Sum of angles* [7], [8], *Wedge method* [8], *Swath method* [9], *Thin regular slices* [10], *Dual representation* [11], *Sign of offset* [12] and *Cell Based Containment Algorithm* [13]. Among them, Cell Based Containment Algorithm (CBCA) was considered the most suitable method for online processing with large polygon shape datasets [13]. It belongs to the class of algorithms using pre-processing that means the map is organized into some segments, and the location of the checked point in any segment will decide whether it is in a polygon or not. Every further checking requires $O(1)$ time complexity only.

Manuscript received February 27, 2013; revised May 16, 2013.

The authors are with Vietnam National University, Hanoi, Vietnam. (e-mail: sonlh@ vnu.edu.vn).

Despite the fact that CBCA is the effective algorithm for large polygon shape datasets, it can be ameliorated further through some modifications in the border cells determination process. Our contribution in this paper is a novel containment test algorithm that integrates a new procedure to specify the border cells with parallel computing. It will be compared with CBCA by experiments to verify the efficiency. This method will also be applied for the terrain query in a 3D WebGIS system, aiming to retrieve the attribute information related to a terrain.

The rests of this paper are organized as follows. Section 2 briefly introduces CBCA method of Zalik. The novel method CBA will be presented in Section 3. The evaluations of computational complexity and experiments will be given in Section 4. Section 5 presents an application of CBA for the terrain query. Finally, we will make conclusions and future works in the last section.

II. Overview of CBCA Method

CBCA method has two main parts: i) Partitioning the polygon into a uniform grid; ii) Inclusion test for a given point.

In the first part, a uniform grid is set up to cover the polygon. This grid contains a suitable number of cells that balance the time for the rasterizing process and the quality of inclusion test afterward. Indeed, the number of cells is determined by a simple heuristic such as,

$$NoOfCells_x = 2\left\lceil \frac{x_{\max}-x_{\min}}{y_{\max}-y_{\min}}\sqrt{n} \right\rceil, \tag{1}$$

$$NoOfCells_y = 2\left\lceil \frac{y_{\max}-y_{\min}}{x_{\max}-x_{\min}}\sqrt{n} \right\rceil, \tag{2}$$

where n is the number of polygon vertices and $x_{\max}$ ($x_{\min}$) is the maximal (minimal) values of data points in the dimension x. Likewise, $y_{\max}$ ($y_{\min}$) is the maximal (minimal) values of data points in the dimension y_{mi}. After the grid is created, the cells containing parts of the polygon border are determined as *Grey*. A sub-procedure so-called Code-based Algorithm, which in essence is an improvement of Bresenham method [14], was used for this task. It stores all cells in one way connected list and uses Bresenham algorithm to find Grey cells. A comparison between two neighbouring Bresenham cells is then performed to detect the outliers. Final results are all Grey cells that consist of polygon's border lines.

The last step in the Initialization phase is to specify the inner cells (*Black*) and the outer ones (*White*). The authors used an improvement of the classical raster-based algorithms [7] for this task. The algorithm starts by moving from the borders of the grid in left, right, up, and down directions until a Grey cell is encountered. All the traversed cells are marked as White. The flood-fill algorithm of Foley et al. [7] is then applied to detect the outliers in the unclassified cells. Four rays from the middle point of the unclassified cell are sent in four directions; left, right, bottom and top until they meet the first White cell. The ray, which crosses the smallest number of cells, is accepted for the final estimation. The number of intersections between the chosen ray and the edges stored in Grey cells is counted by Ray Crossing method [6]. If the number of intersections is even, the cell has the same color as the cell at which the ray stops. Otherwise, the color is inverse. The flood-fill algorithm stops when all cells are marked.

In the second part, the inclusion test specifies the cell in the grid containing the tested point q and returns the result following by the cell's color.

III. Case Based Algorithm

Throughout the previous section, we recognize that the first part of CBCA can be ameliorated by incorporating parallel computing with a new procedure of border cells determination. Specifically, after the grid is formed as in the Initialization phase of CBCA, we divide all polygons into some processors or computers of a parallel system. Polygons are sent to the processors following by their identification codes (ID) one after another. The merging process in the Master processor will combine all different grids in all processors into a unique one by the following criteria:

- If the cells (i, j) in all grids contain the zero and a positive value, the value of this cell in the final grid is the positive one. This number is the ID code of a ubiquitous polygon. The cell (i, j) is equivalent to the Black cell in this situation.
- If all cells consist of the zero and a negative value, we adopt the negative one as the final value of the cell.
- If many negative values are found in all cells, we record all these ones. In this case, a Grey cell containing many border lines of some polygons is recognized.
- The cells, consisting of zero values only, will make the value of the final cell become zero.

Ex. 1: Let us see the Fig. 1 and Fig. 2. These figures describe the marking results in two processors. In Fig. 1, these are three polygons with the ID codes from one to three. Their orientations are (+) (Def. 1). The inner cells are marked by the ID codes of polygons. The outer cells are assigned the zero values. The border cells are given by the negative values of the ID codes. Some cells are assigned more than a value, i.e. the cell (6, 6) in Fig. 1, where many border lines of polygons cross over. Fig. 2 shows the marking results in the second processor. In Fig. 3, the merging result is presented. For example, the values of cell (14, 5) in two processors are two and zero. Thus, the final value of cell (14, 5) in Fig. 3 is two.

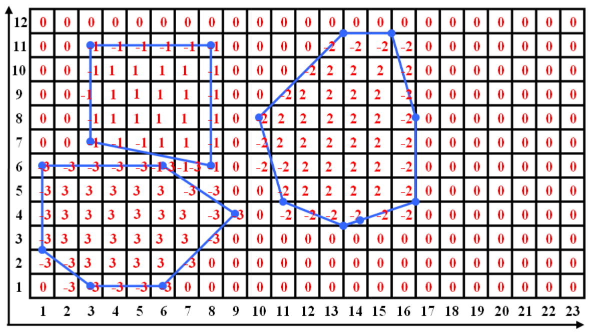


Fig. 1. The marking results in the first processor

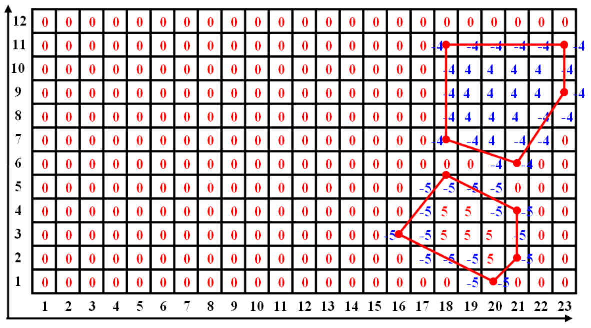


Fig. 2. The marking results in the second processor

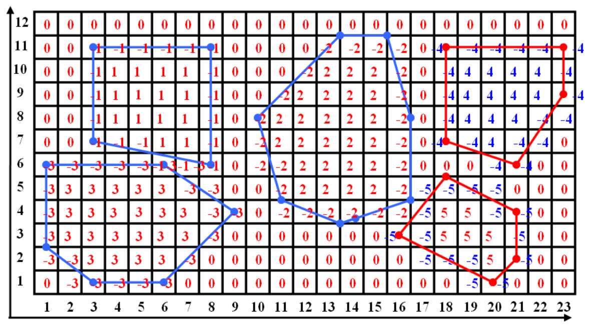


Fig. 3. The merging results in the Master processor

The marking process in a processor is similar to that in the first part of CBCA method. Its output is a grid whose cells are marked by ID code of the polygon that crosses over them. However, in order to enhance the performance of the algorithm, we use a new procedure to specify the border cells through some possible cases in the relation between the polygons and the grid. This procedure is integrated with the process of specifying the inner and outer cells in CBCA method, and is shown below.

Step 1: For each polygon $U_j = < M_1^j, M_2^j, .., M_{n_j}^j, Dt^j >,\ j = \overline{1, l}$, find the cell in the grid containing $M_i^j = \left(M_i^j.x, M_i^j.y\right),\ i = \overline{1, n_j}$ whose four grid's nodes are:

$$\left\{\left(X_i^j, Y_i^j\right), \left(X_i^j + s_x, Y_i^j\right), \left(X_i^j, Y_i^j + s_y\right), \left(X_i^j + s_x, Y_i^j + s_y\right)\right\}, \tag{3}$$

$$\left(X_i^j, Y_j^j\right) = \left(x_{\min} + \left[\frac{M_i^j.x - x_{\min}}{s_x}\right] \times s_x,\; y_{\min} + \left[\frac{M_i^j.y - y_{\min}}{s_y}\right] \times s_y \right), \tag{4}$$

$$\left(s_x, s_y\right) = \left(\frac{x_{\max} - x_{\min}}{NoOfCells_x}, \frac{y_{\max} - y_{\min}}{NoOfCells_y} \right). \tag{5}$$

The $[\;]$ symbol refers to the truncation function to an integer value. However, if one of the following conditions is true, finding the cell containing the point M_i^j is ignored, and we turn to the next one.

$$\left[\frac{M_i^j.x - x_{\min}}{s_x}\right] = \frac{M_i^j.x - x_{\min}}{s_x}, \tag{6}$$

$$\left[\frac{M_i^j.y - y_{\min}}{s_y}\right] = \frac{M_i^j.y - y_{\min}}{s_y}. \tag{7}$$

Step 2: Mark these cells by the negative value of the ID code of polygon U_j.

Step 3: If the line connecting two consecutive vertices of the polygon $\left(M_i^j, M_{i+1}^j\right)$, $i = \overline{1, n_j - 1}$ does not lie on the grid's edge, mark all cells in the intersection between it and the grid by the negative value of the polygon's ID code.

- If the line intersects the vertical grid's edges, two left and right cells of the intersection points will be marked.
- Otherwise, two up and down cells will be marked if the line crosses the horizontal grid's edges.
- If the intersection points are the grid's nodes only, we ignore the line and turn to the next one.

Step 4: Handle some special cases of the polygon's vertices as follows.

- *Case 1*: If the intersections between a polygon's edge and the grid are the grid's nodes only, we mark all the cells in the diagonal through the middle points. For example, from two consecutive vertices $\left(M_i^j, M_{i+1}^j\right)$, $i = \overline{1, n_j - 1}$, we find a set of grid's nodes $\left\{\left(X_i^j, Y_i^j\right)\right\}$. A middle point between two consecutive grid's nodes or between a grid's node and a polygon's vertex is found, such as,

$$\left(T_x, T_y\right) = \left(\frac{X_i^j + X_{i+1}^j}{2}, \frac{Y_i^j + Y_{i+1}^j}{2} \right). \tag{8}$$

The cell containing this point can be totally determined as in Step 1. Similarly, we find other cells and mark them by the negative value of the polygon's ID code.

- *Case 2*: A grid cell may contain many polygon's vertices. If it was marked beforehand in Step 2, we do nothing. Otherwise, mark the cell.
- *Case 3*: Some polygon's edges lie on the grid's edge. In this situation, the orientation Dt^j of the polygon U_j is utilized. This case can be solved by Dt^j and the right-hand rule that means we will mark the cells which are pointed out by the perpendicular vector of the polygon's edge.

Step 5: Grey cells are totally marked. We then use the method in CBCA to specify the inner and outer cells. These cells are marked by ID code of polygon and zero value, respectively.

Step 6: Store these marked values of the polygon, and clear all values in the grid. Repeat these steps for other polygons in the processor.

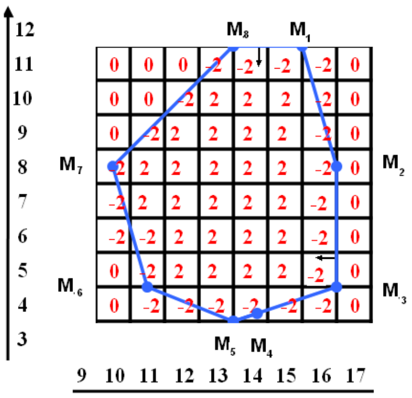


Fig. 4. An example of CBA algorithm

Ex. 2: An example of ID code determination process can be seen in Fig. 4. This figure is the enlargement of the polygon whose ID code is two in the first processor (Fig. 1). The polygon contains eight vertices. Step 1 of the above procedure will eliminate the grid's nodes and the points that lie on the grid's edges. Indeed, only the cells (14, 4), containing M_4, and (10, 8), containing M_7, are marked after the Step 2. Except the polygon's lines $\left(M_2, M_3\right)$, $\left(M_4, M_5\right)$ and $\left(M_8, M_1\right)$, Step 3 will mark all the cells in the intersection between the polygon's edge and the grid, such as the cells (16, 8), (16, 9), (16, 10) and (16, 11) with the line $\left(M_1, M_2\right)$. In Step 4, Case 1 works with the line $\left(M_7, M_8\right)$. All the cells in the diagonal, i.e. (10, 8), (11, 9), (12, 10), (13, 11), will be marked. Case 2 will look for the cell (14, 4) where two polygon's vertices M_4, M_5 are contained. Nevertheless, this cell was marked in Step 1. Thus, nothing is performed in this case. In Case 3, the line $\left(M_2, M_3\right)$ lies on a vertical grid's edge. As mentioned before, the orientation of this polygon is (+). Thus, the perpendicular vector points out to some cells, such as cell (16, 5), (16, 6), (16, 7) and (16, 8). These cells are then marked by the negative value of the polygon's ID code. Similar marking process will be applied for the line $\left(M_8, M_1\right)$. All Grey cells in this polygon have been marked already. The inner cells, i.e. cell (15, 8), and the outer cells, i.e. (17, 9), are marked by the ID code of polygon and the zero value, respectively through Step 5.

The computational time of the pre-processing part will be reduced by the integration of parallel computing with the border cells determination procedure in the *Case Based Algorithm* (CBA). This helps accelerating the overall process and will be suitable for our considered context.

IV. Evaluations

A. Time and Space Complexity

Forming the grid requires $O(n\times l)$ time complexity where n is the number of vertices in a polygon, and l is the number of polygons in a shape dataset. The division of the polygons into the processors and the merging process takes $O(NoOfCells_x\times NoOfCells_y\times l)=O(n\times l)$. In a processor, we have to specify the border, inner and outer cells of a polygon and number all of them. Step 1, 2 and Case 2 of Step 4 of this process requires $O(1)$ time complexity. Step 3 and Case 1, Case 3 of Step 4 obtains $O(NoOfCells_x+NoOfCells_y)\approx O(\sqrt{n})$ time complexity in the worst cases and $O(1)$ in the best ones. Step 5 requires $O(n\times\sqrt{n})$. The number of polygons assigned to a processor is $[l/k]$. Thus, the total complexity in a processor is $[l/k]\times O(\sqrt{n}+n\times\sqrt{n})$. Therefore, the overall time complexity of CBA is $O(\max\{n\times l;[l/k]\times[\sqrt{n}+n\times\sqrt{n}]\})$. The total space complexity of CBA is $O(n\times l)$.

B. Experimental Setup

We have implemented the proposed algorithm (CBA) in MPI/C programming language and executed it on a Linux Cluster 1350 with eight computing nodes of 51.2 GFlops. Each node contains two Intel Xeon dual core 3.2GHz, 2GB Ram. The experimental results are compared with those of CBCA algorithm, which has been recompiled to run in the same configurations with ours. The experimental data are taken from the Bolzano - Bozen province, including a vast of benchmark polygon shape datasets [15].

C. CBA vs. CBCA by the Number of Vertices

In this section, we compare the computational times of two algorithms in the pre-processing part following by the number of vertices of a polygon. Fig. 5 describes their serial times. Fig. 6 highlights the comparison of computational times of CBA using one, two, three and four processors. Results in Fig. 5 show that the serial computational time of CBA is slower than the one of CBCA when the number of vertices is smaller than 15821. However, CBA is faster than CBCA for the remains. For example, when the number of vertices is small, i.e. 22, the computational times of CBA and CBCA are 5.469 and 0.62 seconds, respectively. These times in cases of a medium number of vertices, i.e. 33023, are 27.79 and 40.9 seconds, respectively. For the large number of vertices, CBA is still faster than CBCA, for example 50.14 and 179.17 seconds in case of 94853 vertices. On average, the computational time of CBA is faster than that of CBCA by 2.31 times.

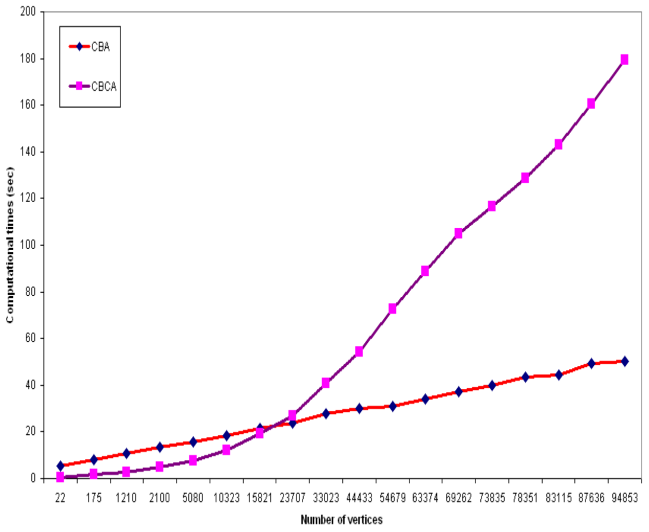


Fig. 5. The serial computational times of CBA and CBCA (sec)

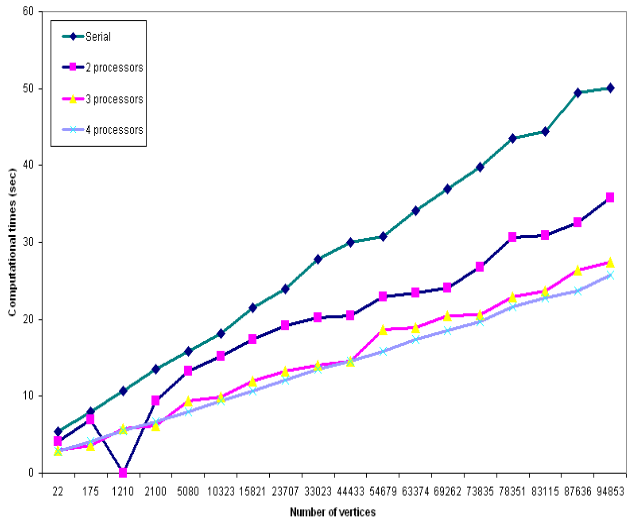


Fig. 6. CBA with many processors by the number of vertices

The average processing time per vertex of CBA is 0.00062 seconds while that of CBCA is 0.0015 seconds. This means that it takes CBA around 0.00062 seconds to process a vertex in the polygon shape. This number is smaller than that of CBCA. Thus, more vertices are provided, the difference between two algorithms is getting obvious. When the number of vertices is very large, i.e. 94853, this difference is maximal. Through the figure, we can recognize that the amplitude of these lines is expanded following by the increment of the number of vertices. Obviously, the modification of the border cells determination process in CBA has ameliorated the computational times of CBCA as shown by the experimental results in Fig. 5.

In the following experiments, we aim to find the answer for the question: "Can we reduce the computational time of CBA significantly with the support of parallel computing?". Although the serial computational time of CBA is faster than that of CBCA, the processing times of CBA, especially in cases of a large number of vertices, are still large. For example, it takes CBA approximately 50 seconds to process

94853 vertices. Consider our problem that runs CBA over the web environment with a large number of vertices, it is necessary to speed up the whole algorithm. In this situation, the use of parallel computing is undeniable.

In Fig. 6, the results of running CBA algorithm with one, two, three and four processors are illustrated. Obviously, more processors are used, less computational times are expensed. For example, the serial processing time of a small number of vertices, i.e. 22, is reduced by 25.4% when using two processors, 44.97% with three processors and 48.91% with four processors. Similarly, these numbers in cases of a medium number of vertices, i.e. 33023, are 27.11%, 49.43% and 51.37%, respectively. For a large number of vertices, the reducing percents are 28.64%, 45.39% and 48.69%. On average, using two processors will reduce the serial computational times by 25.67%. Using three and four processors will make larger reduction by 46.61% and 49.71%, respectively. The processing time of the case above (94853 vertices) with four processors is now 25.7 seconds only. Obviously, the support of parallel computing helps accelerating the whole process, and makes the deployment of CBA over the web environment become reality.

In what follows, we will find the most suitable number of processors for CBA algorithm. Because some additional computations such as communication cost and synchronization have to be paid, large number of processors is sometimes not as effective as the medium or small ones. In order to find the suitable number of processors, we use the speed up and efficiency. The speed up is defined as $S = T_s / T_p$, where T_s (T_p) is the serial (parallel) computational time, respectively. The efficiency is determined as $E = S / k$, where k is the number of processors. Results are depicted in Fig. 7 and Fig. 8.

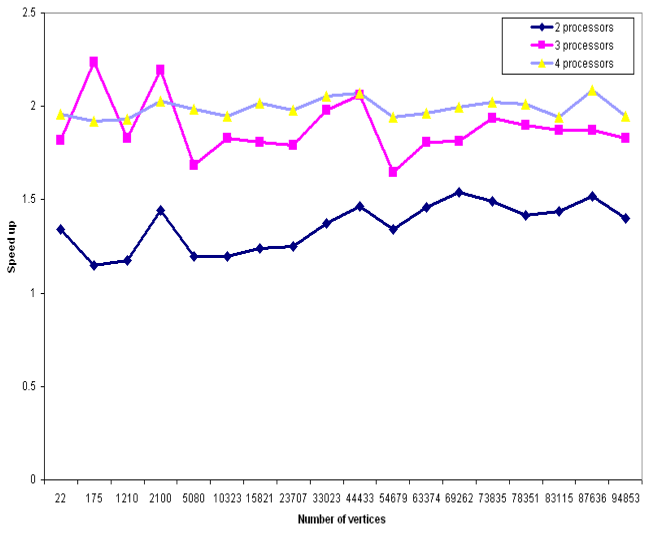


Fig. 7. The Speed up

Through these figures, we can recognize that the speed up of using three processors is nearly double the one of two processors. However, the speed up value of four processors is approximately the one of three processors. Thus, we may predict that three or four processors is the critical state that means using more than this number of processors will not increase the value of speed up, but even reduce it. In Fig. 7, the "3 processors" line is better than the "4 processors" one when the number of vertices is smaller than 3000. For the remains, the "4 processors" obtains the best values. Therefore, we need to check the results in Fig. 8 for the final decision. This figure shows that the efficiency of "4 processors" line is the best among all other ones. Indeed, the suitable number of processors is four.

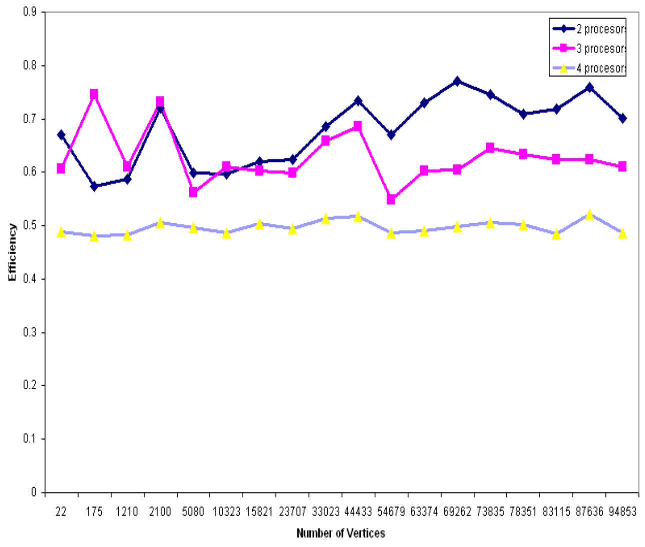


Fig. 8. The Efficiency

Some major conclusions extracted from this part are shown below.

- The modification of the border cells determination process makes CBA faster than CBCA.
- Parallel computing, especially with four processors, helps CBA deploy over the web environment.

D. CBA vs. CBCA by the Number of Polygons

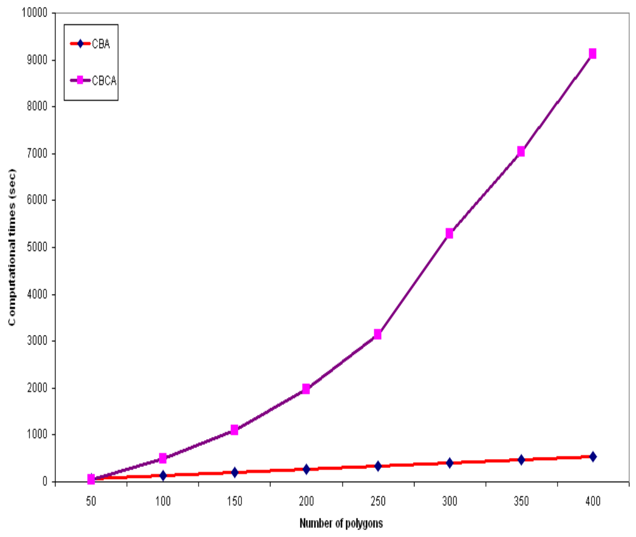


Fig. 9. The comparison of CBA and CBCA by the number of polygons

In Fig. 9, we compare the computational times of CBA and CBCA in the pre-processing part following by the number of polygons. The results show that CBA is slower than CBCA when the number of polygons is small, i.e. below 50. The computational times of CBA and CBCA in this case are 65.99 and 40.25 seconds, respectively. However, CBA is faster than CBCA when the number of polygons increases.

For example, when the number of polygons is 100, the computational times of CBA and CBCA are 129 and 491 seconds, respectively. Similarly, these times are 534 and 9213 seconds when the number of polygons increases to 400. Because the average time to process a polygon of CBA is faster than that of CBCA namely 1.33 and 14 seconds, the computational time of CBA is indeed better than that of CBCA. Its effectiveness is getting obvious when a large number of polygons are processed.

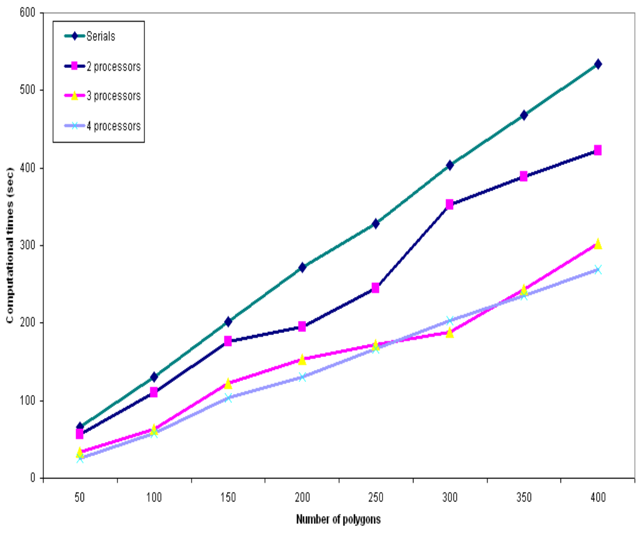


Fig. 10. CBA with many processors by the number of polygons

Due to the parallel computing between processors, more polygons in a shape will lead to more effectiveness of CBA. Fig. 10 illustrates the computational times of CBA when using from one to four processors following by the number of polygons. Similar results with Fig. 6 are obtained in this situation. The computational time of CBA using two processors is 1.23 times faster than the serial one. Similarly, these values in cases of three and four processors are 1.87 and 2.01, respectively. These numbers are smaller than those in Fig. 6 since processing many polygons requires additional times to store and clear marked values as in the Step 6 of the ID code determination process. The processing time for a large number of polygons, i.e. 400, with four processors is 269 seconds. It is 1.98 times smaller than the serial one. Therefore, we can recognize that the computational cost is reduced remarkably with the support of parallel computing, even in the cases of large numbers of polygons. Some calculations about the speed up and efficiency values of those times are also calculated. Indeed, the suitable number of processors for CBA method is four.

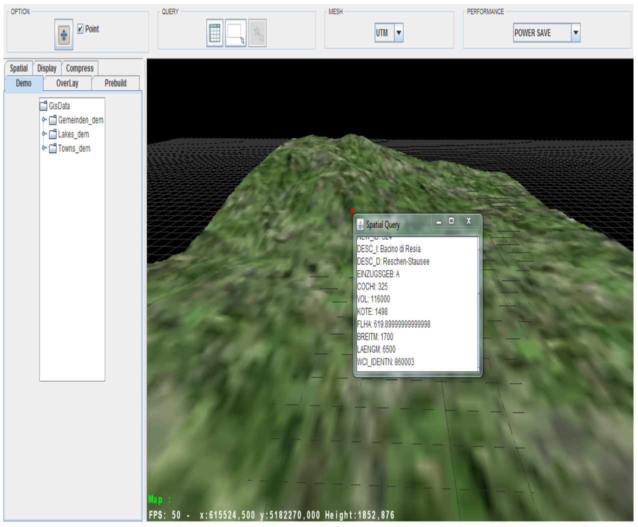
Through this part, we also obtain the same conclusions with the previous one.

V. An Application of CBA for the Terrain Query

In this section, we will illustrate an application of our method to the terrain query. We know that terrains are the input of the 3D WebGIS. There are some standards represented for the terrain. Among them, *Digital Elevation Model* (DEM) is the most commonly used due to its simplicity and ease to access. According to Albani et al. [16], DEM consists of a matrix data structure with the topographic elevation of each pixel stored in a matrix node. DEM is distinct from other representations such as TIN and contour based data-storage structures. It is generated by many methods, for example, via satellites, air planes, LIDAR technology, etc.

Most 3D WebGIS systems do not support the attribute query on a terrain. They often consider the terrain as a normal three-dimensional object that is used to model the Earth with the higher precision and detailed level than previous ones, instead of performing some exploitation in order to extract meaningful information and knowledge, serving for the decision-making process afterward. In [17], the authors stated that the most important point to differentiate between a GIS and a cartographic system is the information inside each region, i.e. the coordinates and the location's name. They allow us to comprehend the internal characteristics of regions in order to make some spatial analyses and increase the additional values of the map. Traditional GIS provides attribute query as a basic function. Therefore, the attribute query on a terrain of the 3D WebGIS is a must.

The main reason to make the attribute query on a terrain less interest turns out to be the structure of the DEM terrain. As mentioned above, this kind of terrain contains the spatial information about elevation values only, and related attribute data are not attached; thus making the query impossible. Therefore, some additional sources should be given in equivalent to the DEM terrain. They have to contain both the attribute records and spatial data such as the vector or raster. In this situation, the polygon shape dataset (Def. 2) is the most suitable one. Assume that we have a couple of data: the DEM terrain and the polygon shape dataset. While the shape dataset is used to represent a ubiquitous thematic map, the DEM terrain is the three-dimensional representation of this map that means each elevation value in DEM always attaches to a coordinate (x, y) in the shape. This kind of data is widely used in most of the mixed 2D-3D GIS applications, such as in COMGIS project [15]. Thus, the terrain query is possible in this context.



DESC_I: Bacino di Resia
DESC_D: Reschen-Stausee
EINZUGSGEB: A
COCHI: 325
VOL: 116000
KOTE: 1498
FLHA: 619.89999999999998
BREITM: 1700
LAENGM: 6500
WCI_IDENTN: 860003

FPS: 50 - x:615524,500 y:5182270,000 Height:1852,876

Fig 11. Terrain query implementation in a 3D WebGIS system

The basic idea is to change the queried object from a DEM terrain to its polygon shape dataset. Because we have a relation between an elevation value in the DEM terrain and the coordinate (x, y) in the polygon shape, the query can therefore be performed on the polygon shape by ignoring the z value in the DEM terrain. The terrain query is now changed to the traditional containment test. Additionally, this kind of query often requires fast processing as we are working on the web environment, and thousands of requests can occur at a same time. As such, our method CBA can be utilized in this situation.

Some modifications of CBA method for the terrain query should be performed. In the Initialization step of CBA method, the grid is built by a heuristic method. However, it should be replaced by the grid of DEM terrain in order to match the coordinates. The number of grid's nodes, in this case, is even larger than the previous one since the resolution of DEM terrain is high. Thus, it brings more accurate results for the end-users. Nevertheless, the cost of computation is increased as a result. The supports of parallel computing in CBA are really effective in this case.

We have implemented the procedure above for the 3D WebGIS system in COMGIS project [15]. The results are shown in Fig. 11. When users click a point on the terrain, attribute information such as terrain's name, area, etc. will be displayed. This function is convenient to quickly determine the information related to a terrain, serving for further advanced mining techniques.

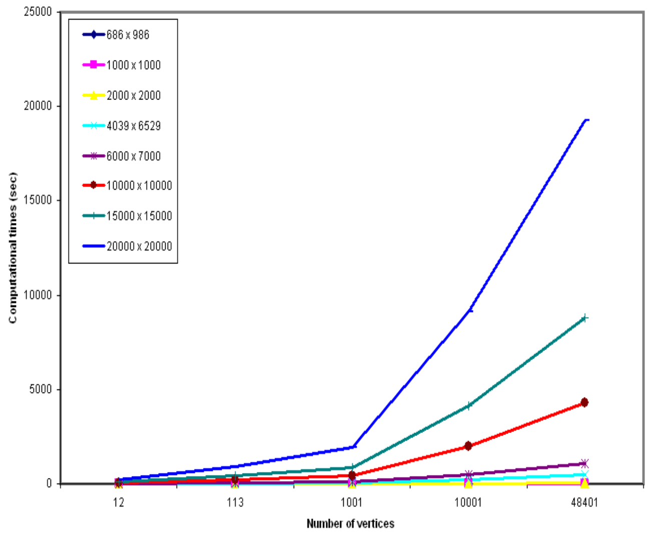


Fig. 12. The computational times of the procedure (sec)

We also made some experiments to check the procedure in this 3D WebGIS system. Fig. 12 shows the serial computational times of the procedure when processing DEM terrains with different sizes, i.e. 686 × 986, and polygon shape datasets having different number of total vertices. The results show that there is not clear difference between some sizes of the DEM terrain when the number of vertices is smaller than 113. The maximal processing time in this case is around 3.5 minutes when working with the large DEM terrain (20000 × 20000). The minimal time is 70 milliseconds on the small DEM terrain (686 × 986). The difference between them is 3.4 minutes. On average, the time to process a small number of vertices is around 45 seconds. The difference is getting obvious when more vertices are added. In the case of 48401 vertices, the maximal and minimal processing times are 5.343 hours and 7.1 seconds, respectively. The difference between these times and the average time are 5.341 and 1.17 hours. We can recognize the distinct difference when comparing these values with the ones above. The average times to process a vertex with small (686 × 986), medium (6000 × 7000) and large DEM terrains (20000 × 20000) are 0.002, 0.32 and 5.73 seconds, respectively. Thus, small DEM terrains often run faster than other ones.

These remarks help us to predict the computational time of the terrain query procedure on a specific DEM terrain. Besides, we should choose the small (medium) number of vertices and DEM terrains for the sake of fast computation over the web environment.

VI. Conclusions

In this paper, an improvement of the containment test algorithm CBCA that works with polygon shape data in WebGIS applications was presented. It integrated a new procedure to specify the border cells and the parallel computing technique to CBCA method, aiming to enhance the performance of pre-processing process. Numerical results showed that this method is better than CBCA, especially when processing a large number of polygons in the shape dataset. It was also applied to the terrain query on a 3D WebGIS system. Some analyses about its performance were made for the better understanding of the proposed algorithm.

Future researches will investigate some advanced mining methods on attribute information and applications of our method in other problems.

Acknowledgment

The authors are greatly indebted to the Editor-in-Chiefs Prof. Abdul Razaque and anonymous reviewers for giving us valuable comments to improve the clarity and quality of the paper. This work is sponsored by the research grants of National Foundation for Science and Technology Development (NAFOSTED - Project No. 102.01-2012.14) and two major projects of Vietnam National University, Hanoi, entitled "Study about some data mining methods in Geographic Information Systems" and "Research on spatial data clustering algorithms and applications".

References

[1] M. Chen and P. Townsend, "Efficient and consistent algorithms for determining the containment of points in polygons and polyhedra," in *Proceedings of Eurographics'87*, Elsevier Science, Amsterdam, 1987, pp. 423–437.

[2] F. Feito, J. C. Torres, and A. Urena, "Orientation, simplicity, and inclusion test for planar polygons," *Computers & Graphics*, vol. 19, no. 4, 1995, pp. 595–600.

[3] M. Gombosi and B. Zalik, "Point-in-polygon tests for geometric buffers," *Computers & Geosciences*, vol. 31, 2005, pp. 1201–1212.

[4] C. W. Huang and T. Y. Shih, "On the complexity of point-in-polygon algorithms," *Computers & Geosciences*, vol. 23, no. 1, 1997, pp. 109–118.

[5] J. Lia, W. Wang, and E. Wu, "Point-in-polygon tests by convex decomposition," *Computers & Graphics*, vol. 31, 2007, pp. 636–648.

[6] U. Manber, *Introduction to algorithms: a creative approach*. Reading, MA: Addison-Wesley, 1989.

[7] J. D. Foley *et al.*, *Computer Graphics, Principles and Practice, 2nd edn.*. Reading, MA: Addison-Wesley, 1990.

[8] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction, 2nd edn*. New York: Springer, 1985.

[9] K. B. Salomon, "An efficient point-in-polygon algorithm," *Computers & Geosciences*, vol. 4, no. 2, 1978, pp. 173–175.

[10] V. Skala, "Point-in-polygon with O(1) complexity," Technical Report No, University of West Bohemia, Pilsen, Czech Republic, 1994, pp. 68-94.

[11] V. Skala, "Line clipping E2 with suboptimal complexity O(1)," *Computers & Graphics*, vol. 20, no. 4, 1996, pp. 523–530.

[12] G. Taylor, "Point in polygon test," *Survey Review*, vol. 32, 1994, pp. 479–484.

[13] B. Zalik and I. Kolingerova, "A cell-based point-in-polygon algorithm suitable for large sets of points," *Computers & Geosciences*, vol. 27, 2001, pp. 1135–1145.

[14] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM System Journal*, vol. 4, no. 1, 1965, pp. 25–30.

[15] L. H. Son, P. H. Thong, N. D. Linh, T. C. Cuong and N. D. Hoa, "Developing JSG Framework and Applications in COMGIS Project," *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, 2011, pp. 108-118.

[16] M. Albani, B. Klinkenberg, D. W. Andison,and J. P. Kimmins, "The choice of window size in approximating topographic surfaces from Digital Elevation Models," *International Journal of Geographical Information Science*, vol. 18, no. 6, 2004, pp. 577–593.

[17] S. Rana and J. Sharma, *Frontiers of Geographic Information Technology*. Netherlands: Springer-Verlag, 2006.

Le Hoang Son is a researcher at the Center for High Performance Computing, VNU University of Science, VNU. His major field includes Soft Computing, Geographic Information Systems and Parallel Computing. He is a member of IACSIT and also served as a reviewer for some international journal.

Nguyen Duy Linh is a researcher and Master student at the Center for High Performance Computing, Hanoi University of Science, VNU. His research interests include Geographic Information Systems and Grid Computing. Email: linhnduy@gmail.com. Tel.: +84-904-641-190.

Nguyen Thi Hong Minh is a doctor and vice dean of School of Graduate Studies, VNU. Her major researches include Parallel Algorithms and Molecular Dynamics Simulation. So far, she has performed many important, major projects of VNU, especially in interdisciplinary fields. Email: minhnth@vnu.edu.vn. Tel.: +84-904-101-065.