Some Issues of Information Storage Management for GIS Applications on Pocket PC and Windows CE 3.0

Duong Anh Duc, Le Thuy Anh, Do Lenh Hung Son
* Dept of Software Engineering, University of Natural Sciences, National University of HCM City,Vietnam
Tel : +84-08-8354-266 Fax : +81-00-111-6789 E-mail: {daduc, ltanh, dlhson}@fit.hcmuns.edu.vn

Abstract:

The Pocket PC has become more popular in market because of the advantages of its small size and convenience for regular customers. Pocket PC is a mobile device so that we can receive the benefits of shared data over a wireless network. Enabling us to transmit data to a central location, simply messaging from one point to the next, its ability to share information across a wireless platform is becoming central to our communication needs.

However, using Windows CE – an embedded operating system, as well as being designed for mobile users, there are many limitations to memory and speed of arithmetic operations on Pocket PC. As a result, developers have to deal with many difficulties in managing information storage when developing applications, especially Geography Information System (GIS) applications. In this paper, we propose some efficient methods to store GIS data and to increase the speed of displaying maps in GIS applications developed on Pocket PC and Windows CE 3.0.

1. INTRODUCTION

As Vietnam's economy is currently experiencing the process of development and integration, traffic has become one of the primary concerns of the society. However, the traffic system of the country in general and in Hochiminh City in particular is still facing many problems and thus needs amelioration, such as improving the public transportation system and providing vehicle operators with sufficient information.

Besides, the increasing number of foreign tourists to Vietnam calls for an effective method of providing them with the city's geographical information, and the digital map is the best solution in this case.

Given the above results, we have done researches and developed a digital map software on some embedded devices, especially on Pocket PC and Windows CE devices.

PocketPC runs on embedded operating system Windows CE, which provides limited memory to applications developed on it. Therefore, when developing Geography Information System (GIS) applications on PocketPCs and Windows CE, it is recommended that we optimize the memory usage and enhance the speed of these applications.

In this paper, we present some solutions used widely on the world and our implementation in order to reduce redundant memory and to increase the speed of rendering maps in Pocket PC and Windows CE 3.0.

2. SOME FEATURES OF GIS-APPLICATION AFFECTING TO MEMORY USAGE AND THE SPEED OF APPLICATIONS

2.1. The detail level of rendering spatial objects

Depending on the purpose of using, spatial data need a particular accuracy. For example in a GIS land management system, a lake is represented by a 350-points polygon, but in a GIS traffic system, 110 points are enough.

2.2. Topology Relationship level

As mentioned in 2.1 – topology network, we need to choose a suitable topology level because beside the storing cost, processing cost for every time updating data will be huge and it is wasteful if the topology level is larger than necessary.

2.3. The detail level of attribute data

Storing unnecessary attribute data will increase the storing cost but the processing cost may not increase if well management

2.4. Calculated attributes

Some attributes such as length (or perimeter), the minimal bounding rectangle (MBR)... can be calculated from spatial data. Precalculating these attributes and updating only when there are changes will make the processing cost come-down considerably but increase storing cost. Besides that, in some basic GIS softwares, that programmers can alter directly to spatial data may lead to the asynchronism of the calculated attributes and spatial objects. The strong point of this method is the ability to query these attributes and to make the best of competence of DBMS.

2.5. Real number storing and computation

All operations on spatial data are real-number operations. In principle, processing cost on real numbers is larger so far than on integer.

Table 1 Time required (ms) for real and integer number computation

	Devices	Pentium II 333 MHz	Pentium IV 1.5 GHz	Pocket PC 2002 Emulator (on P4 1.5 GHz)[1]	Pocket PC HP Jornada 928 (ARM4 206 MHz)
+/-	int	3.845	2.157	3.600	11.794
	float	3.716	2.125	6.750	85.538
	Double	3.705	2.171	2.175	241.039
×/÷	int	17.465	9.000	9.013	46.090
	float	13.790	4.672	4.900	116.092
	Double	6.419	4.641	4.925	399.845

Table 1 shows that the time for processing floating point number on Pentium is not longer than integer. With integer multiplication and division, if the numbers are stored as real, the processing cost of multiplying and dividing is really smaller.

While the processing time between integer and real numbers is not different so far on Pentium-computer, table 1 also shows a considerable difference on Pocket PC: 7.25/20 times for +/- and 2.5/8.67 times for ×/÷ when using 4/8 bytes floating point numbers instead of integer.

Should we replace real-number processing with integer processing?

Depending on the type of processor (floating point unit) that our application will run on, we will decide to replace floating point operators or not. Another element should be mentioned is that the limitation in data range of integer operators may lead to overloading, especially with integer multiplication.

3. HIGH-COST OPERATIONS IN THE POCKET PC GIS APPLICATIONS

The property of GIS is that every step of processing or analyzing needs to deal with a large amount of objects.. We have to deal with many big problems, especially in devices not having a powerful processing unit like desktop. This section shows some processes of these problems. Because these processes are used frequently, it is essential that we consider to improve the speed when running in embedded devices.

3.1. MBR (Minimal Bounding Rectangle)

MBR is used frequently in GIS to save the cost of redundant clipping steps. We can use MBR to make an index for accessing spatial objects when making queries such as: specifying objects to draw in a map, finding objects next to each other, specifying map objects when users click on it. Figure 1 shows the way to use MBR to completely reject the cost of drawing objects not in the display area including the cost of clipping and mapping objects onto the display area.

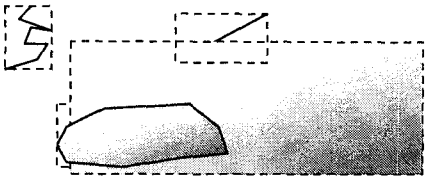

Fig. 1. MBR is used to specify whether an object can be rendered.

MBR is used in almost all operations on spatial objects. Therefore when objects are defined with too many points, it makes the cost of specifying MBR raise too high.

3.2. Clipping

Clipping is an opertion that is used very often in GIS applications. When we need to display or print a part of a map, we must use clipping to draw parts of the objects that stay in the display area only. Figure 2 shows the clipping operation before rendering objects on the screen.

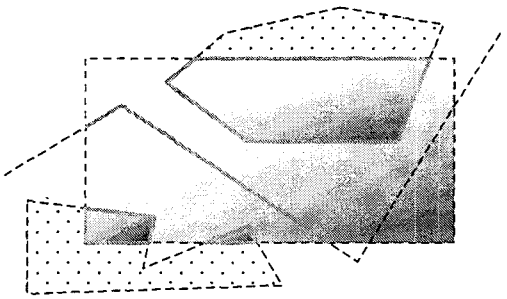

Fig. 2. An special case of using clipping.

We also need to pay attention to some special cases when implementing clipping such as: clipping a polygon can result in one polygon or more as shown in figure 2.

The cost of clipping is very high, thus we should try to use MBR in GIS applications whenever the output result is NULL to avoid clipping. The cost of testing whether two MBR rectangulars intersect each other is certainly lower than the cost of clipping.

3.3. Rendering spatial objects

Spatial objects are saved in database with their real co-ordinates. In order to render these objects onto the screen, we have to make a transformation to change

from the real co-ordinate to the screen co-ordinate. We must use a corresponding affine transformation matrix depending on the display area and screen resolution of the users.

The number of GIS applications that allow users to rotate display area freely is so rare. If there were any, it'd only allow users to rotate at specific angles such as: 90°, 180°, 270° ... Supplying users with a rotating display area leads to the complexity of using clipping to reject the objects staying outside the display area, because the edges of the MBR are no longer parallel to x-axis and y-axis.

Thus, the transformation from real co-ordinates to screen depends on 3 parameters: the ratio (s), the distance x (dx), the distance y (dy) according to the formula below:

Table 2 The transformation from real co-ordinates to screen co-ordinates

$$x' = round(s \times x) + dx$$
$$y' = round(s \times y) + dy$$

With:

x,y: real co-ordinate
x',y': transfomred co-ordinate
s: the ratio
dx, dy: the distance

In some special cases we also need to transform from screen co-ordinate to real co-ordinate such as when users click on the screen to make spatial queries, we have to specify the real co-ordinates by using the formula below:

Table 3 The transformation from screen co-ordinates to real co-ordinates

$$x = \frac{x' - dx}{s}$$
$$y = \frac{y' - dy}{s}$$

3.4. Hit Test – Specifying chosen spatial objects

Theoretically, specifying chosen objects is simply finding the object having the shortest path to the point where users click on. For many objects such as: zigzag, polygon, or even the area made up of a polygon including many polygons inside...the cost will be very high.

In fact, when users choose an object, it is not necessary to pay attention to other objects too far from the clicking point (larger than a number d). Therefore, we can use a square with edges of $2d$ to reject all the objects that stay outside the selected area by testing the MBR property of these objects. Frequently after doing this, the number of objects needing to be calculated the distance from the clicking point becomes so small.

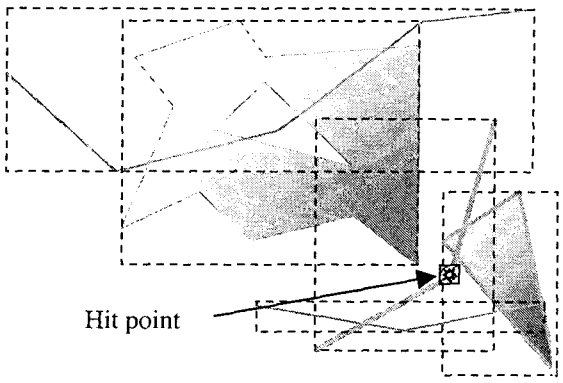

Fig. 3. Using a square in specifying hit point.

4. SOME EFFICIENT METHODS TO ACHIEVE HIGH PERFORMANCE IN THE POCKET PC GIS APPLICATION

4.1. Spatial index tree for MBR

Whenever we want to find out which objects will be rendered on the screen or whenever users want to make non–spatial queries, we need to test if the MBR of these objects intersects the rectangular of the screen, as well as testing to reject objects that we don't need to render immediately.

If we use an index for MBR, the cost will be lower than the cost of testing MBR of all objects.

Here, we show some implementing approaches of spatial index that we have tested their performance on Pocket PC.

4.1.1. Common approaches

Binary Search Tree

The specific characteristic of this method is that searching-space will be divided into two parts in honrizontal and vertical continuously until there is no part left. Some examples: KD-Tree, LSD-Tree...

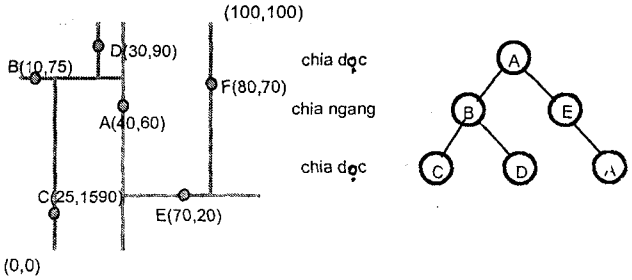

Fig. 4. The structure of KD-Tree

Binary Search Tree in cooperation with B-tree. Today there are several algorithms such as: KBD-Tree, R-tree, R*-Tree, TV-Tree, X-Tree... with various approaches.

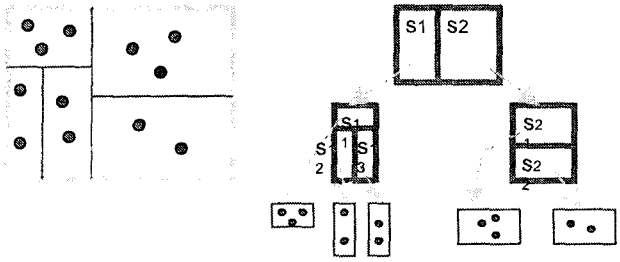

Fig. 5. The structure of K-D-B- Tree

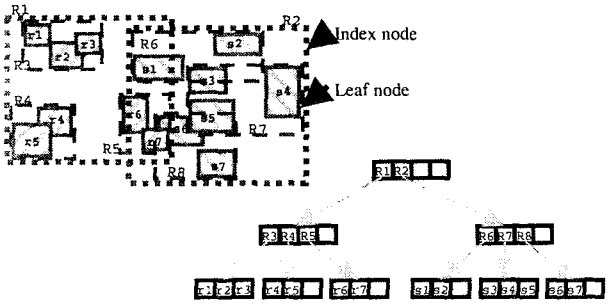

Fig. 10. The structure of R-tree

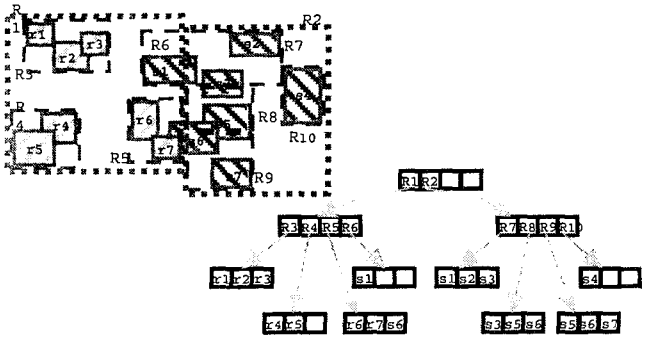

Fig. 11. The structure of R*-tree

4.1.1. Our own approach

When using R-Tree, in some cases we'll have to examine two sub-trees because the MBR of two these may intersect each other. When using R*-Tree, the objects may appear in the two sub-trees, thus we have to mark the objects to avoid re-exmanining them.

We suggest using R+Tree to implement and set the condition in that one object can only appear in one sub-Tree (like the condition of R-Tree). If there is an object appearing more than once, we'll only save it at the lowest node of the tree having a path to that object (in the worst case this is the root of the tree).

The way of this division is to divide searching-space horizonly and then vertically: the node at the odd level will be divided vertically and the node at the even level will be divided horizonly. After dividing the searching space into two parts, there will be some objects appearing on the border (these objects belong to both parts). In this case these objects will be saved at the node being traversed.

One node can be stored up to bucket objects, if there are other objects inserted, we have to re-divide the searching space. The value of *bucket* we recommmend here is **8** because if bucket is smaller than 8 the cost of dividing and searching is high, if bucket is bigger than 8 the Sequential Search can not be applied.

The defect of this method is that when we divide the searching space, there will be more than *bucket* objects appearing on the border. At that time, the node must be stored more than *bucket* objects. This will lead to a more complicated data arrangement, moreover the searching process can be uneffective because the node has too many objects, especially the root of the index tree.

After creating a tree, all spatial queries will be made as below: *Examine(root, rectangular-want-to-test).* Function *Examine(node, rectangular)* tests whether the rectangular intersects the MBR or not. If there is no intersection between them, fucntion ends. Otherwise, the function will examine all the objects served in the node sequentially, and then call recursively to the two sub-node:

- If the left node is NOT equal NULL then Examine(left-node, rectangular)
- If the right node is NOT equal NULL then Examine(right-node,rectangular)

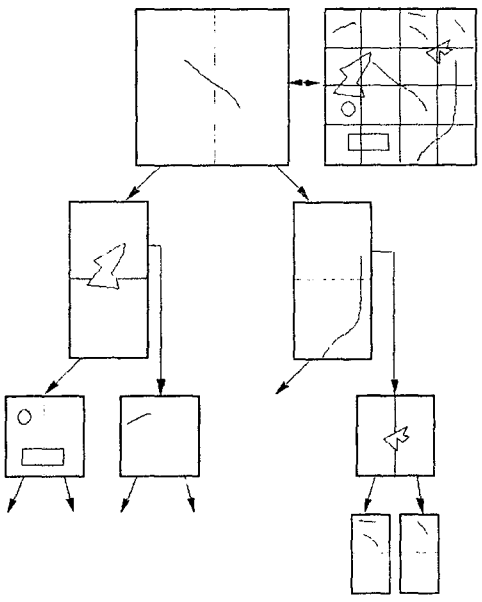

Fig.1 Tree division with bucket = 2

4.2. Reduction in Real number multiplication

When rendering objects on the screen, we need to transform from the real co-ordinate to the screen co-ordinate with 3 parameters *s,dx,dy* (see the formula in section 1.1.3.4)

Note that when users have chosen an apropriate display angles, all the operations to examine the map will almost be movement operations. This operation only makes the paramater *dx, dy* changed, the parameters *s* is left unchanged. Thus, we can reduce the cost of multiplying real numbers by using secondary variables. Here are the code in C++:

```
int dx, dy;
float s;
Point
SpatialObject::Transform(TFPoint
fpointRealWorld)
{
    Point pointResult;
    pointResult.x       =       Round(s*
       fpointRealWorld.x) + dx;
    pointResult.y       =       Round(s*
       fpointRealWorld.y) + dy;
    return pointResult;
}
```

In this code, when s unchanges, the result of function *Round(s * fpointRealWorld.x)* can be saved in an integer variable nScaledX for every point of the object. The same thing for integer variable nScaleY. These two variables will be updated whenever s changes or the space informations of the objects change.

```
nScaledX            =           Round(s*
fpointRealWorld.x);
nScaledY            =           Round(s*
fpointRealWorld.y);
```

At that time, the transformed fucntion can be rewritten as follow:

```
Point
SpatialObject::Transform(TFPoint
fpointRealWorld)
{
        Point pointResult;
        pointResult.x   =   nScaledX   +
dx;
        pointResult.y   =   nScaledY   +
dy;
        return pointResult;
}
```

In this function, we only use two real number additions, so we have reduced two multiplication and two steps to round real number. Therefore, the limitation of processing real numbers are put away.

5. CONCLUSION

We have proposed some methods used to get the good result in information storage management in order to increase the speed of rendering maps in Pocket PC and Windows CE 3.0. These methods, as mentioned, have showed its advantages in reduction the cost of manipulation. Our own experiments also showed that these is efficient in some other embedded devices.
Even if the memory capacity (the object store) of the Pocket PC has increased recent years, these methods is also worth to be considered because its ability to achieve surprisingly high performance.

References

[1] Nicholas R. Chrisman (1997), *Exploring Geographic Information Systems*, University of Washington.

[2] National Air and Space Measum (1998), *GPS: A new constellation*[2].

[3] Ajenar B Kartika, Singgih Supriyanto, *GIS Database Design and Presentation*, IndoMap.

[4] Lê Thụy Anh- Đinh Bá Tiến, *The traffic guide system*, Bachelor Thesis, Dept of Software Engineering, University of Natural Sciences, National University of HCM City,Vietnam, 2000.

[5] Võ Sỹ Nam – Đỗ Lệnh Hùng Sơn, *Digital Map on Pocket PC*, Bachelor Thesis, Dept of Software Engineering, University of Natural Sciences, National University of HCM City,Vietnam, 2003.

[2] http://www.nasm.si.edu/galleries/gps/