# Dual-cache Structure Based Large Scale Texture Mapping for Real-time Terrain Rendering

Dong Tian
Image Processing and Pattern Recognition Laboratory
Beijing Normal University
Beijing 100875, P.R. China
tiandong@mail.bnu.edu.cn

Xiaodong Wang
Image Processing and Pattern Recognition Laboratory
Beijing Normal University
Beijing 100875, P.R. China
xd_wang@mail.bnu.edu.cn

Xin Zheng
Image Processing and Pattern Recognition Laboratory
Beijing Normal University
Beijing 100875, P.R. China
zhengxin@bnu.edu.cn (Corresponding Author)

***Abstract*—There are two key problems in efficient large scale texture mapping for terrain rendering— efficient data organization and real time data updating in memory. In order to solve these problems, in this paper we propose a quadtree based indexing method to organize multi-resolution images and to fast retrieve data from disk; For memory updating, we present a real time dual-cache structure based updating method, which effectively reduces the frequency of data refresh. We also innovatively use a wavelet image enhancement algorithm to enhance original terrain texture, which obtain richer edge information and give us a more realistic effect in terrain rendering. Through the analysis of storage efficiency and rendering speed of our experiment, this dual-cache structure based method solves rendering speed and memory limit problems perfectly.**

***Keywords*—Large scale texture mapping; Dual-cache structure; Image enhancement; Quadtree; Virtual reality**

## I. INTRODUCTION

Virtual reality technology [1] is more and more important today. Real time terrain rendering is one of the key problems in this area. In order to obtain more real effect, high resolution satellite texture image has been used in many virtual environments. But the data size of these images is very large and exceeds the loading ability of the RAM in PC and graphics cards. We have to piece the large image into smaller patches and load one patch each time. But when the viewport is far away or high and more virtual scene comes into our field of view, this method can not deal with the huge data. In 2004, Hoppe, and others proposed a new geometry-clipmap[2] method for large scale terrain-rendering, which generated geometry LOD[3, 4] as the same way of texture Mipmap[5] and cut interested regions on every level [6, 7, 8]. It is an effective solution in the memory limitations of terrain data loading. But as the texture data significantly more huge than terrain's, we still need a more effective way to solve the problem of texture processing.

There are two key problems in efficient large scale texture mapping for terrain rendering. One is searching a rapid and effective texture organization and indexing way in the disk. Because terrain texture image stored in disk is usually tremendous. The scene should be dynamically refreshed while viewpoint moves and texture retrieval is a continuous process, so we need a fine data organization and indexing method for facilitate data retrieval with low frequency of data IO. Texture updating method in RAM and VRAM directly affects the display efficiency, so it is another key problem in efficient large scale texture mapping. By now the two problems mentioned above haven't been satisfactorily resolved. In order to solve the first problem, in this paper we propose a quadtree based indexing method to organize multi-resolution image data and to fast retrieve data from disk; For memory updating, we present a real time dual-cache structure based updating method. By this way, texture remains part of quadtree structure in the memory and then we generate the second cache by cutting invalid edge [9], which effectively reduces the frequency of data refresh. In the process of generating multi-resolution texture structures, some match artifacts between different layers of texture image are introduced for the reason of detail lost. In order to debase these artifacts we also enhance original texture's edges innovatively to obtain richer edge information by use a wavelet image enhancement algorithm, which give us a more realistic effect in terrain rendering. Through the analysis of storage efficiency and rendering speed of our experiment, dual-cache structure method solves rendering speed and memory limit problems perfectly.

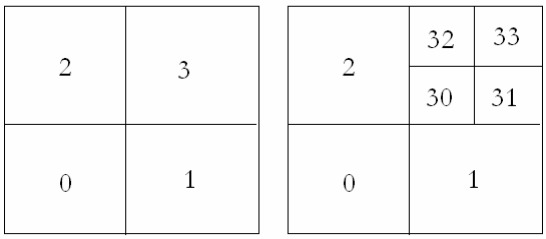
The organization of this paper is as follows: In the next section, we will give the achievement of quadtree based indexing technology used in texture preprocess. In section 3, we will present our dual-cache structure based updating

method in real time rendering. In Section 4, we will give the image enhancement effect by using wavelet method to our work. In section 5, we will give some result of our experiments and last give the conclusions and works in the future.

## II. QUADTREE BASED INDEXING

### A. *Quadtree Encoding*

Quadtree code is a kind of encoding method which is used to designate a particular block in the overall texture.




(a) (b)

Figure 1. Quadtree encoding

Figure 1 (a) shows the coding rule of quadtree, the texture size of $2^n$ were divided into four child nodes and then encoded them as shown in figure, thus we get four simple quadtree codes 0,1,2,3 in anti-clockwise. Then we divide each node and encode them with the same rules, to which we add its father node's quadtree code, as child node 3 shown in figure 1 (b). Repeating this operation, we get all quadtree codes of the original texture images, as shown in figure 2.

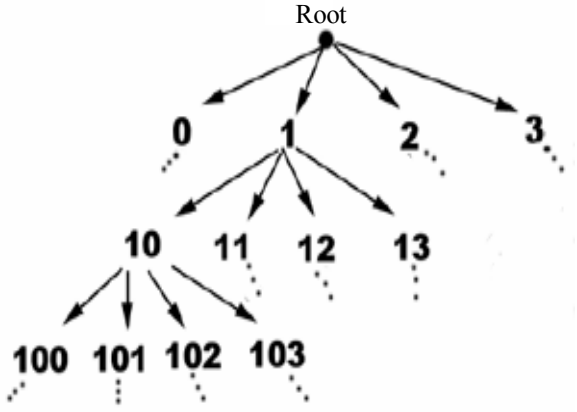



Figure 2. The structure of the quartree codes

This code reflects the position of texture region in the overall texture picture, and we can get the current texture region's father region code quickly by its quadtree code. If the length of the current texture region's code is n, its father region's code is its first n-1digits.

### B. *Texture Preprocess*

The data of the original texture is tremendous and can not be read into the memory in one time, therefore need a preprocess for data organization, which including two parts : Down sampling and Segmentation.

Down sampling [10] is used to get lower resolutions of the original texture images. We use the simplest sampling method-getting the average of four adjacent pixels as a new pixel value. One down sampling process will get half resolution of the texture. Repeating this operation, we get a serial of textures, which buildup a texture pyramid and are encoded with 0 1 2 3 …from the original one.

Segmentation is to cut the textures in the texture pyramid into some small patches, whose size helps to both read the data into memory one time and do not waste system bandwidth.

We organized the textures patches with quratree code and save them in the disk with name of clip[-a][-b], where clip is the original texture's name, a represents the level of the pyramid this texture patch belonging to and b is the texture patch's quadtree code in layer a.

## III. DUAL-CACHE STRUCTURE BASED TEXTUR UPDATING

We build up a dual-cache structure in order to debase the frequency of data exchange both from disk to memory and from memory to video memory.

### A. *Exchange Texture Data between Memory and Disk*

Because of memory limit, we assign a small viewport around viewpoint. On each layer of texture pyramid，only few (up to 4) texture patches near to the viewpoint can be read into the memory. With the movement of the viewport, we must update data from external memory (disk) to the interior memory. But the reading way just relying on the quadtree's father-child relationship does not meet our demands. As shown in figure 3, the current viewpoint drops on the right bottom corner of the texture patch encoded with 300. According to the quadtree's father-child relationship, we should read patch 300, 301, 302 and 303 into the memory. Now we will face a problem: the texture we read into the memory can not cover current viewport (red box in figure 3). At this time, the texture we should to read is 122,123,300,301. And only by this way, we can avoid the range of the viewport beyond the texture in memory.

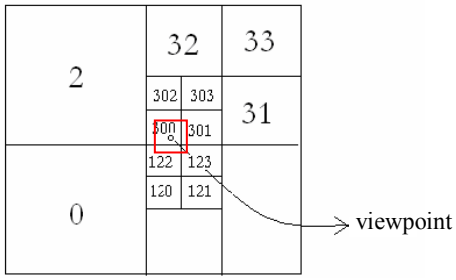



Figure 3. Read textures from the Disk

Next, we give the method we used to obtain the disk texture patches in accordance with the current view port:

First, we define a structure named "fourimagestruct" as following, which is used to store the texture data in the same level that is read from the hard disk

```
struct fourimagestruct{   //used to storage data from disk
                          // coordinate of current storage area
        GLfloat           x;
```

```
    GLfloat        y;
    BYTE *         data;
};
```

Where, "data" stores the texture data consisted by four hard disk texture patches, (x, y) is the coordinate of bottom right corner of the current texture in the overall texture. Second we calculate the quadtree code of texture patche on lower level on which the viewpoint dropped. This code is used to determine which four texture patches we should read.

As shown in figure 4, ABCDEFGI is nine adjacent texture patches in the disk. For example, if the viewpoint drops on the E-district, thus E-district was divided into four sub-regiones E1, E2, E3, E4, and we get the quadtree code "choosecode" which is deeper than E's quadtree code, then "choosecode" helps to decide which four adjacent regions should be read. If viewpoint drops on E1, we should read four patches DEAB; if on E2, we read four patches EFBC; If on E3, we read four patches GHDE, if the E4, read four patches HIEF. After identifying four texture patches which should be read, we can easily know their quadtree codes by our new method: as is shown, the code of E can be calculate by removing the final value of the code "choosecode". Next we determine the other three patche's quadtree code according to E's quadtree code and the position of E in the four patches.

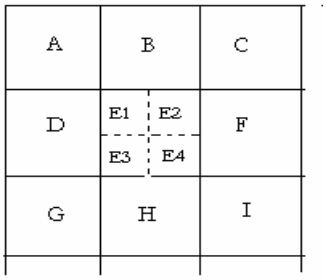



Figure 4. Loaded patches choice on disk

According to the symmetry of patches we can easily get the other three patches' quadtree codes: First identify the final value of the four patches' code, then from the order of which we deduce their father patches' relations. The father patches' relationship can be summarized in the following four situations as shown in figure 5.

From the four texture patches' father textures' relationship, we can get the last second digit of the four patches' Quadtree code, and then we can continue deduce the relationship of the higher level of texture in quadtree with the second four, and then we get the last third values. Repeating this processing, we can get the whole quadtree code of the four texture patches. Plus four texture's level number we can obtain the file name of the four texture patches on the disk which will be read into memory. The most crucial step in this method is how to get the relation of the father textures from the child textures' quadtree code.

After deductions, we find that all the cases of the value of four adjacent texture patches in quadtree code have 16 kinds of style as shown in figure 6.

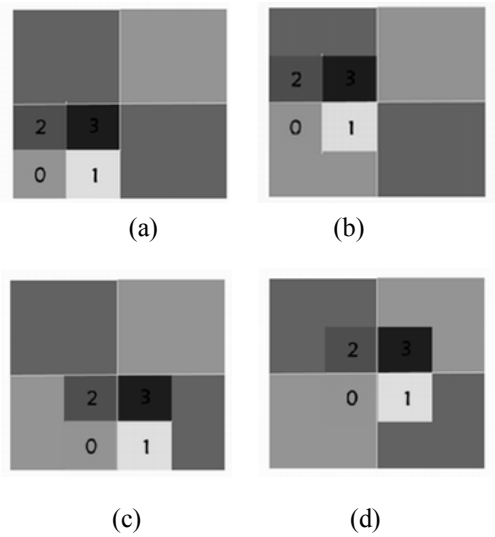



Figure 5. (a) Four child texture patches 0123 belong to the same father texture patche. (b) Four child texture patches 0123 belong to two up-down different texture patches. (c)Four child texture patches 0123 belong to two left-right different texture patches. (d)Four child texture patches 0123 belong to different texture patches.

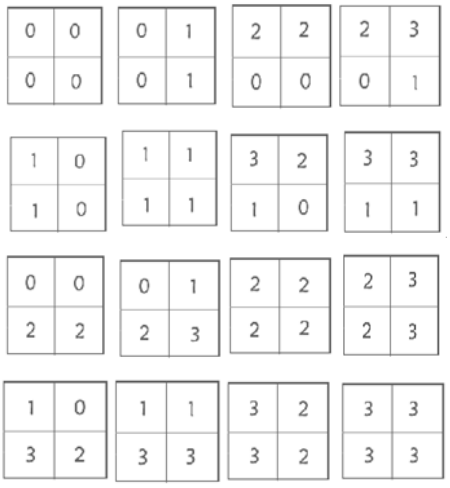



Figure 6. 16 Distributions

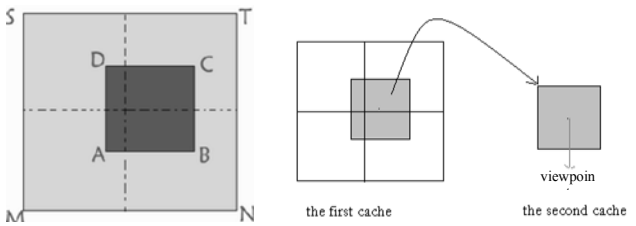
As shown in figure 6, we find that in one row, the values in the bottom left corner of the units are same, but the values in the top right corner are different; in one column, the values in the top right corner of the units are same, but the values in the bottom left corner are different. So the 16 units can be divided by the value of the bottom right and top left corner values.

Finally we relate each unit to a map in figure 5, and set up the following response:

- Figure 5 (a) counterparts in the case of zero
- Figure 5 (b) counterparts in the case of one
- Figure 5 (c) counterparts in the case of two
- Figure 5 (d) counterparts in the case of three

### B. *Texture Updating in the memory*

While the viewpoint moves, the texture in the memory need to be updated in real-time, but the frequency of updating will seriously affect the speed of rendering. For this problem, we propose a dual-cache structure. The first cache stores layers of texture which are read from the disk, the texture size is greater than the size of the viewport; and the second cache memory stores layers of textures cut from the first cache with size equal to the viewport. So, a short-distance moving of the viewpoint will not result in that the viewport exceed the scope of texture stored in the first cache, particularly on lower resolution layers. We just need update the second cache from the first cache's data. Only if the viewport is out of the scope, we reread data from the disk. So most timely, the viewport can slide on the texture in the first cache while the viewpoint moves. It will greatly avoid frequent data exchanging between the hard disk and the memory and improve the rendering speed.




(a) (b)

Figure 7. (a) Position of the viewport (b) Dual-cache structure

As is shown in Figure 7, MNST is first cache constituted by four texture patches, ABCD is the scope of the viewport, which can slides on MNST. The data in the viewport is also stored in the second cache, it will be bound and read into the video memory.

### C. *Video Memory Updating*

Each movement of the viewpoint will result in the data updating of the viewport, this will greatly reduce the efficiency of the system, so here we apply L-district updating technology to update texture in video memory.

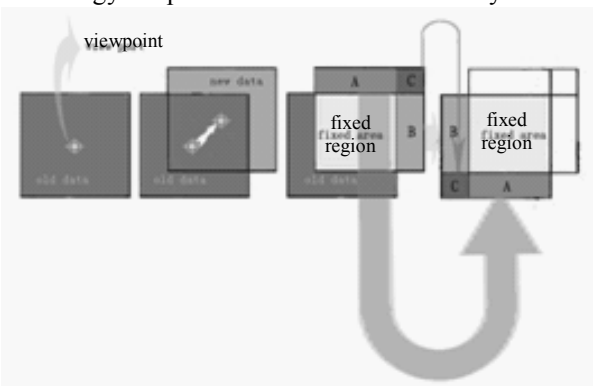



Figure 8. L-district updating process

As shown in Figure 8, the rectangle window is a texture with size same to the viewport stored in the video memory. If the viewpoint changes, the data of the texture memory needs to be updated. Obviously, there is a common region between the old data and new data (the fixed region), this region does not change both before and after the updating, and just three regions A, B, C change, so we just change the data in L area as we need update the texture. This processing is shown in the third and fourth steps in figure 8, when the viewpoint moves to the right corner, data of regions A, B, C in step 3 must be updated. We copy A's texture data to the bottom region, copy texture data of B to the left region, copy the data of C which locate in the top right corner to the bottom left corner. Thus we complete the data updating of the video memory.

## IV. IMAGE ENHANCEMENT

In the texture mapping process, the use of multi-resolution technology lead to some loss of the details in terrain texture and introduce image blur and some match artifacts between different layers of texture image. We applied a wavelet-based image enhancement method [11] to enhance the texture before preprocess, which can increase the details of the texture, Figure 9.(b) shows the original texture with two different resolutions(left part is half resolution of right part). We enhanced the lower resolution part in this image as shown in figure 9.(a). We can find that the blurred image in (b) looks more clear.

(a)

(b)

Figure 9. (a) Enhanced texture (b) Original texture

## V. EXPERIMENT

By using Visual C + + and OpenGL, we establish a 3D navigating system by using our method on a PC (Pentium (R) D 2.8GHz CPU, 1.00GB memory, NVIDIA 256MB video card.) DEM used to be tested is size of 1202 x 802, texture size

is 10000 × 6667. Each pixel occupies three bytes and viewport is size of 256 × 256(pixels). We establish cache 1 with seven layer and cache 2 with seven layer. All the storage space requires: 256 × 256 (texture pixels) × 7 (levels) +512 × 512 (texture pixels) × 7 = 6.5MB, which is much less than Mipmap(shown in Table I). the results of the experiment is: the time of the data exchanging between disk and memory for one time is about 30 ms, memory separate updating time is about 20 ms, L-area technology helps to short the updating time to about 8 ms . As is shown in Table II, The rate of display is about 53 fps.

TABLE I. THE STORAGE SPACE OF MIPMAP AND OUR METHOD

| **Method** | ***Mipmap*** | ***Our Method*** | |
|---|---|---|---|
| | | First Cache | Second Cache |
| | | 5.25M | 1.31M |
| ***Total Space*** | 365M | 6.56M | |

TABLE II. FPS OF SOME VIEWPORT SIZE OF OUR METHOD

| **Window size** | ***Max*** | ***Min*** | ***Mean*** |
|---|---|---|---|
| 256×256(pixel) | 60*fps* | 46 *fps* | 53 *fps* |

Figure 10.(a) shows a mesh terrain which has more than one levels, (b) is the corresponding mapping terrain, the red lines mark different resolution images we used. We give another overall effect of the terrain in Figure 11.

## VI. CONCLUSION

We do not use complicated calculations to complete disk documents' retrieval, but adopt pattern matching method on the basis of pre-generating quadtree codes' fixed model, which greatly increase the speed of retrieval and save time for the disk data's obtaining. Quadtree and dual-cache is used for memory texture 's handling ,which reduce the frequency of data updating effectively. In the early state of the pretreatment, we weighted enhanced the textures, which effectively improves visual effect. The technology fusing double quadtree and dual-cache achieved Clipmap terrain mapping perfectly.

## ACKNOWLEDGMENT

The research work described in this paper was supported by grant from the National Natural Science Foundation of China (Project No.60703070) and grant from the National Natural Science Foundation of China (Project No.60675011)

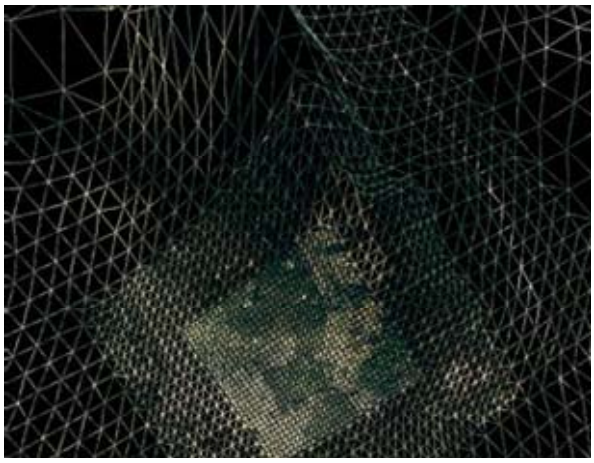
## REFERENCES

[1] Jiazhu Wu、Gang Dang、Huafeng Liu. Visual simulation technology and its application[M]. Xi'an：Xi'an Electronic Science and Technology University. 2001. 213-215.

[2] Frank Losasso, Hugues Hoppe. Geometry clipmaps. terrain rendering using nested regular grids, ACM Transactions on Graphics. New York: Aug 2004. Vol. 23, Iss. 3

[3] Haifei Liu, Deyan Zang, Jianren Chen. LOD terrain simplification and geometric deformation correction based on Quadtree. Mapping and space geographic information, 2006

[4] Zhao Youbing, Zhou Ji, Shi Jiaoying, Pan Zhigeng, A Fast Algorithm For Large Scale Terrain Walkthrough, CAD/Graphics'2001



(a)

(b)

Figure 10. (a) Multi-resolution of terain polygons. (b)With texture mapping

Figure 11. Overall effext

[5] Ying Du, Yuguo Wu, Xiong You. Research of Mipmap Textures for Virtual Global Terrain Environmen. Mapping Science and Technology Journal 2006

[6] Leon Shirman, and Yakov Kamen, A new look at mipmap level estimation techniques. Computers & Graphics ,Volume 23, Issue 2, April 1999, 223-231

[7] Chih-Chun Chen, Jung-Hong Chuang, , Bo-Yin Lee, Wei-Wen Feng and Ting Chiou, Rendering complex scenes using spatial subdivision and

textured LOD meshes, Computers & Graphics , Volume 27, Issue 2, April 2003, 189-204

[8] Zhengliang Huang, The LOD generation of multi-resolution terrain based on the view port, Ship Electronic Engineering,Issue 3, 2007

[9] Jiang, Zhongding , Luo, Xuan Mao, Yandong Zang, Binyu Lin, Hai Bao, Hujun, Interactive Browsing of Large Images on Multi-projector Display Wall System, Springer Berlin / Heidelberg, 2007

[10] Tam!as Frajka, Kenneth Zeger. Downsampling dependent upsampling ofimages. Signal Processing: Image Communication 19 (2004) 257–265

[11] Hai-feng Cui1, Xin Zheng, Wen-cheng Wang, A Wavelet-based Image Enhancement Algorithm For Real Time Multi-resolution Texture Mapping,2006