2012 International Conference on Image, Vision and Computing (ICIVC 2012)
IPCSIT vol. 50 (2012) © (2012) IACSIT Press, Singapore
DOI: 10.7763/IPCSIT.2012.V50.16

Clipmap based Loading and Rendering of Large Terrain Texture

LUO Zhong-kui[+] and DUAN Ming

College of computer science, Sichuan University, Chengdu, China

Abstract. In order to meet the needs of rendering of large terrain, the loading and rendering methods of large-scale terrain texture are studied. A framework of terrain texture real-time loading and rendering is presented based on clipmap and texture blocks. Texture update and texture sampling are researched and the situations of viewer quickly moving and viewer in high are considered. A method of texture garbage collection based on viewer shift is presented to avoid the failure of fetching texture and to release the unused texture quickly. The results show that the method is simple, efficient and can be used in large terrain texture rendering.

Keywords: clipmap; terrain; texture; redering; shift

1. Introduction

With the development of computer graphics, three dimensional terrain rendering is used for many computer applications such as games, flight and other simulations. Terrain texture mapping as an important part of terrain rendering, plays an important role to enhance the visual effect of the terrain. Therefore, studying a large-scale texture loading and rendering algorithm has significant practical means. At present, the terrain texture rendering approach consists of two types: synthetic-based approach and real texture based approach. Bloom[1] describes a way to blend the textures in the frame buffer, this required terrain data to be draw multiple times. Jenks[2] shows a similar results by using programmable shaders in the GPU. Li[3] describes a tile-based texture mapping method that creates a large texture based on a small sample texture. The final texture is obtained through synthesis in these methods. Although the texture generation is very efficient, it cannot represent the real terrain texture. Some papers describe a method that splits the texture into blocks and loading texture by scheduling blocks in real time[4][5](in Chinese). These methods also use texture blocks for rendering. It is need to solve the problem of stitching texture blocks' boundaries. When the view field increases, the texture data of different resolution transfer switching will cause transmission bottlenecks. The clipmap technology [6] was originally developed for SGI Graphics Workstations and required special hardware support that is not present in commodity graphics hardware. Nevertheless, the clipmap can be emulated by programmable hardware with some effort. In this paper, a framework of large terrain texture loading and rendering is presented based on clipmap and texture blocks. A texture garbage collection based on viewer shift is presented for texture scheduling. The update and sampling of texture are also analyzed.

2. Climpmap technology concept

Not every layer of mipmap texture is fully used when the graphics hardware samples from mipmap texture, just part of each layer data is used. The clipmap texture contains a fixed sized subset of a mipmap texture to reduce the storage capacity, as shown in Fig.1. The pyramid represents each layer of mipmap texture. The fixed sized subset is a quadratic region clipped from every layer of mipmap texture, which is called clip regions .They are shaded green in Fig. 1. If the size of a layer exceeds the clip size, the layer is

[+] Corresponding author.
E-mail address: zhongkuilaoda@163.com

stored partly. These layers are referred to clipmap layers. All clipmap layers are called clipmap stack. The layers that not exceed the clip size are stored fully, which are called mipmap stacks. The texture data of each layer in clipmap stack will change with the shift of its clip center. The clip center is the texture coordinates of the viewer in this layer of mipmap texture. The clipmap stores a part of mipmap, so the amount of GPU memory required for a clipmap is linear in the size of the texture, which can be calculated as follows. Suppose the texture size is 2^n x 2^n with a clip size of 2^m, the memory required can be calculated as $4^m(n-m + 4/3) -1/3$ [6].

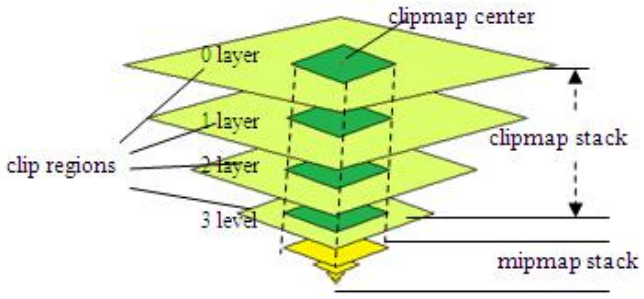

Figure 1. The clipmap concept

3. texture loading and rendering algorithms

The current capacity of computer memory is still limited. In order to load the large-scale texture data, in this paper we present a method that stored the data in the hard disk after splitting texture into blocks and loaded data at runtime through pre-fetching and scheduling strategies. We use a new feature of the graphic hardware called texture array to represent the clipmap stack. The advantage of texture array is that each layer can be addressed with a vector(s,t,p). The first two (s,t) are texture coordinates and (p) refers the layer in the texture array. In rendering, we calculate the appropriate level through the pixel shader to fetch a texel.

3.1. Texture block

The traditional method is splitting a large texture into fixed-size texture block and each block generates its own mipmap texture. This method will cause a large number of disk IO operations when the textures switch between different resolutions. In this article, the method of splitting texture is as follows: first, build all mipmap layers of the texture. Then, split the layers that exceed the clip size into fixed-size texture blocks. For example, suppose a texture size is 16384*16384 and the clip size is 1024*1024. The texture will generate 14 layers of mipmap texture and these layers that are larger than 1024*1024 will be split into blocks.

3.2. Data structure

For each block, there is a TexTile to represent it. Each index of TexTile is kept in a two-dimensional array called TexTileArray2D, TexTile also records its own layer and the index values in X, Z direction. Each layer of mipmap texture has a TileLayer to manage the two-dimensional array. All TileLayers are managed by the TileManager. TileManager logically caches all layers of mipmap texture in main memory. A texture array and a regular mipmap texture are used to represent the structure of clipmap storage. The two textures are managed by VirtualTexture. The mipmap texture always remains the same and the texture array is dynamically updated. VirtualTexture contains an array of clipLayers which record the clip regions in all layers of the mipmap texture. The entire structure is shown in Fig. 2. In order to facilitate scheduling, every TexTile has following states: UNLOAD (not loaded), LOADED (loaded), LOCK (lock).

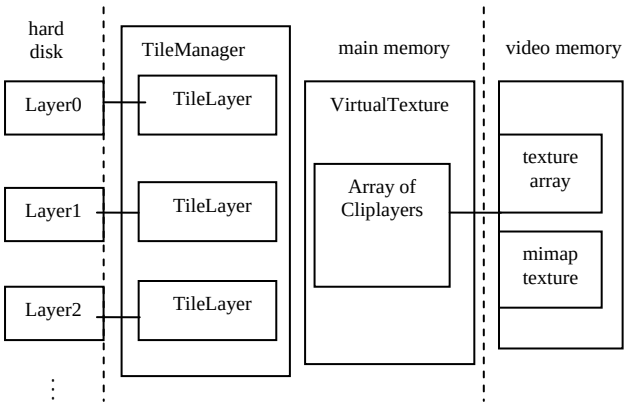

Figure 2. The data structure

3.3. Loading and update of texture

The texture data is loaded into video memory for rendering at last. In order not to block the main rendering thread, texture being loaded into memory from disk is pre-loaded by a background thread. The main thread is responsible for updating data from memory to the video memory and predicting textures blocks. At every frame, the main thread predicts needed texture blocks that may be used in future and puts the blocks into the pre-loaded queue. The background thread is responsible for loading texture blocks in the pre-loaded queue from hard disk into main memory and performing garbage collection to release unused texture blocks. The process is shown in Fig. 3.

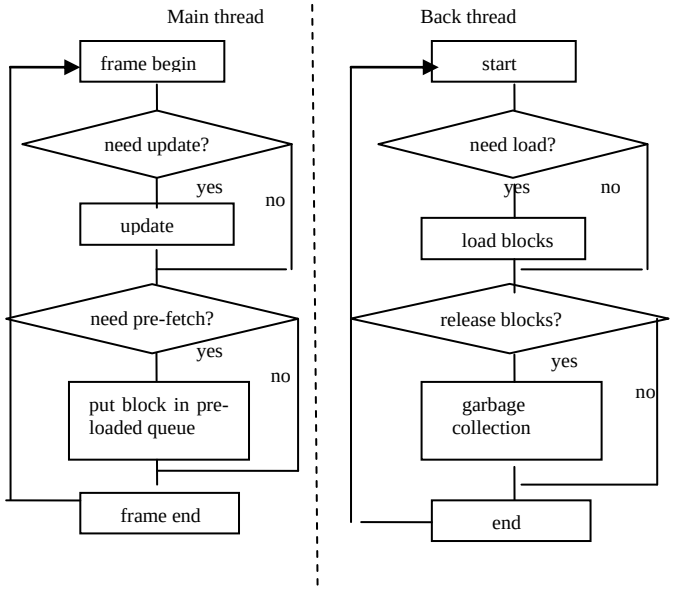

Figure 3. The process of loading texture

Each layer of clipmap will perform the update and the pre-fetch operation. The whole process of each layer is as follows.

1) Calculate the clip center of clip regions in every layer according to the current camera position. Suppose the terrain starts at (X0, Z0) with length and width (W, H), its clipCenter is as follows.

ClipCenter.uv = ((CamPos.x – X0)/W,(CamPos.z – Z0)/H)*MipMapSize; (1)

The MipMapSize is the size of current layer of mipmap texture.

2) Calculate the texture blocks that the clip region covered according to the area of the clip region. Test whether these blocks of texture are loaded into main memory or not. If some blocks are missing, we should load these blocks into the main memory. If these blocks are already in the main memory, we should update the current region of the blocks into video memory. Good scheduling strategy should avoid texture being

directly loaded into main memory from hard disk. In order to reduce the amount of data translated from main memory to video memory, we just update the data of changed region. This can be accomplished by updating the textures of the clipmap layers with tororidal addressing which uses modulo arithmetic to wrap around the edges of the clip region. As shown in Fig.4, the new covered part will coverage to the part that is not used to reduce data to be transferred.

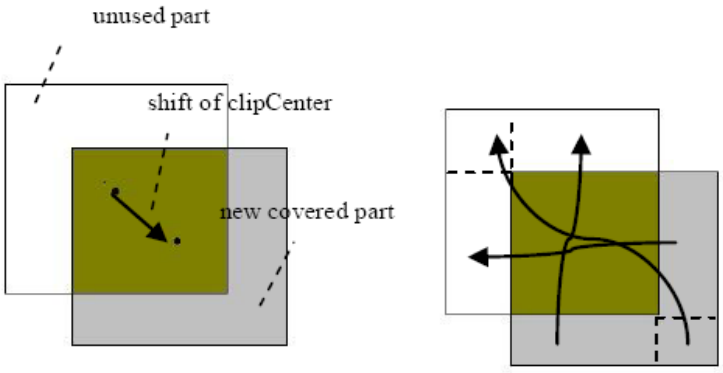

Figure 4. The tororidal update

3) Predict the texture blocks that will be used in future and put them into pre-load queue. The traditional method is adding these blocks that are adjacent to current clip area to pre-load queue. As shown in Fig. 5, the gray part is the current blocks covered by the clip region. The brown part is these blocks to be added to the pre-load queue. This method is not accurate and causes too many blocks to be loaded. The method in this paper is as follows. At the first frame the preloading is as traditional method shown in Fig.5. At every frame we record the clip centers of the current frame and pre frame denoted by (CenterU, CenterV) and (preCenterU, preCenterV). If the line between CenterU and preCenterU crosses the boundary of two blocks in U direction, the preloading is executed in the U direction. The same thing is done in the V direction. As shown in Fig.6, CenterU and preCenterU crosses the boundary of two blocks in the U direction, so blocks of the next row in the U direction will be added to the pre-load queue.

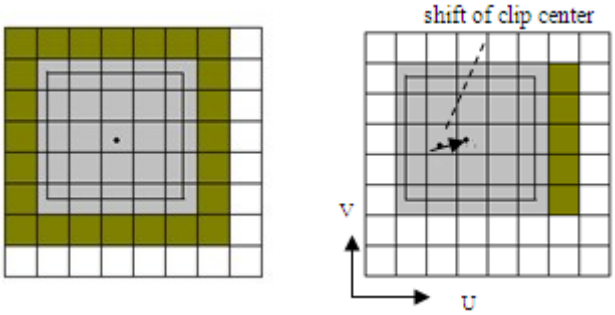

Fig.5. traditional preloading Fig.6. preloading in this paper

The frame rate will drop quickly when the viewer is moving fast. This is due to the large data updated in each frame. The time that the texture pixels stay on the screen is short because of the quickly moving of the viewer. We set a maximum offset value (Maxdu, Maxdv) to avoid too much data to be loaded. If the offset of clip center in each frame is greater than Maxdu or Maxdv, we set the offset to the maximum offset. If the viewer is high in the sky, some layers of high resolution don't update data because the texture is far from the viewer.

3.4. Texture garbage collection

Algorithm of loading blocks is described above. The memory increases when too many blocks are loaded. We should unload some blocks to avoid using too much memory. Those blocks do not need in future should be unloaded. Traditional method is selecting the least recently used blocks to unload. While the LRU algorithm has two disadvantages: 1) failure of fetching the texture blocks; 2) cannot release the unused blocks

quickly. As shown in Fig.7.The rows of A to G are cached in memory. The clip center is in the C row at first. The LRU garbage collection begins when the clip center moves from C to D. The G row will be unloaded because G-row has not been used at current. However, the G row may be used in the future. The A row must wait for a long time to unload until the blocks of A row become the longest unused blocks.

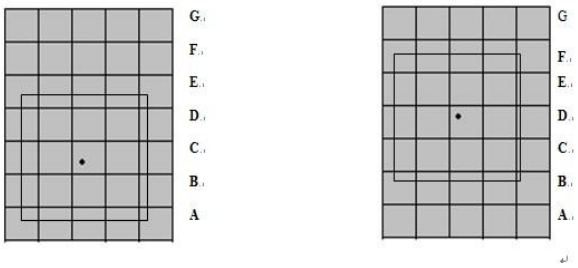

Figure 7. LRU garbage collection

In order to overcome these shortcomings, this paper presents a garbage collection strategy based on the shift of viewer. The main process is as follows.

1) Calculate the offset vector (du,dv) between current and previous clipCenter and normalize it .

2) In order to cache blocks around the clip region, expand the coverage area of clip region.

3) Project the blocks that are not in the area of clip region on the direction of offset vector (du,dv). The unloading is always beginning at the farthest block in the opposite direction of offset vector. As shown in Fig. 8, the brown part is the expanded area of clip region. The gray part is the blocks that may be unloaded .The gray blocks are projected on the direction of offset vector. So the blocks are unloaded from the bottom left to upper right to ensure the garbage collection always begins at the blocks that are far away from the viewer.

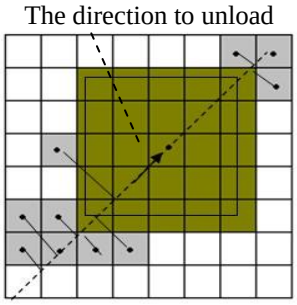

Figure 8. Garbage collection in this paper

3.5. rendering

When rendering the regular mipmap texture, a level must be determined for each fragment that samples the clipmap texture. We determined the preferred level by calculating the rate of change of the texture coordinates in screen space with the partial derivative operators. We calculate the preferred level K as follows.

$K = \log2 (\max (| ds / dx, dt / dx |, | ds / dy, dt / dy |))$ [7]. (2)

The ds /dx represents the partial derivative of texture pixel' coordinate s in screen x direction, ds /dy represents the partial derivative of texture pixel' coordinate s in screen y direction. $| ds / dx, dt / dx | = sqrt ((ds / dx)^2 + (dt / dx)^2)$.

In any case, the preferred level may not be available because the clip region contains a subset of the layer of mipmap texture. It is always possible to get a similar texel value by sampling from a lower level.

The clip region texture data is stored with tororidal addressing. When sampling an edge texel with linear filtering, the nearby texels across the edge are interpolated. The edge texels are not usually similar to their nearby texels across the edges .This will result in a bad look. As shown in Fig.9, the red dot defines the logical edges of the texture. When sampling the yellow texels, the nearby texels across the edge are interpolated. This will results an obvious border. So we prevent sampling across the edges by reducing the clip region extent

with the size of one texel. Texels in the gray part are demoted to a lower level. The texels are always sampling from the white area.

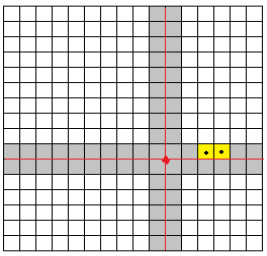

Figure 9. sampling of clip region

4. Resulting

The hardware environment is Intel Core 2 Duo E7400 (2.80GHz) CPU and GTX260 GPU. Software environment is the VS2008 and OpenGL. The Puget Sound terrain texture of size of 16k*16k is used and sup-sampled to 32k * 32k. The block size is 256 * 256. The rendering frame rate fluctuates between 105 and 120 per second. The effect is shown in Fig. 10 and Fig. 11. Fig.10 shows the different clip regions in layers of mipmap texture. Fig. 11 shows the final results. The frame rate is unstable when we use the LRU garbage collection algorithm.

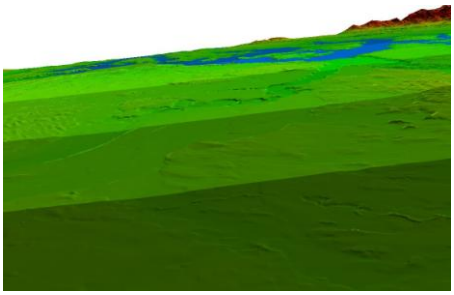

Figure 10. the clip region of texture rendering

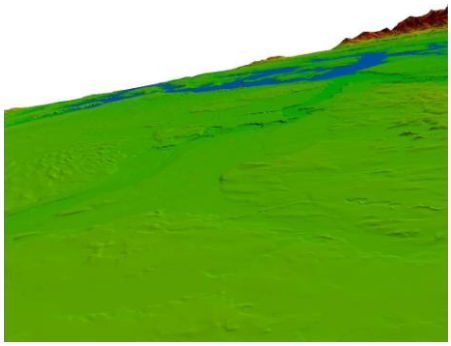

Figure 11. Final result of texture rendering

5. Conclusion

We presented a framework to loading and rendering large terrain texture based on clipmap and texture blocks. An improved method to overcome the shortcomings of LRU garbage collection algorithm is proposed and the result is good.

In the process of terrain texture rendering, we didn't take into account the change of texture data. However, the texture may change due to user interaction. Therefore, future research will focus on the following area: texture loading and rendering with run-time modifications.

6. Acknowledgment

This paper is supported and financed by the National High Technology Research and Development Program (863 Program) of China (Grant No. 2009AA01Z332).

7. References

[1] C.Bloom “Terrain texture compositing by blending in the frame-buffer”, 2000. http://www.cbloom.com/3d/techdocs/splatting.txt.

[2] T. Jenks, “Terrain texture blending on a programmable GPU,” 2005.http://www.jenkz.org/articles/terraintexture.htm.

[3] Li YiWei, “Tile-based texture mapping on graphics hard ware,” in SIGGRAPH/EUROGRAPHICS Conferrence On Graphics Hardware ,pp. 55-63, ACM, 2004.

[4] Zhou Ke, “Massive landscape paging scheduling and reatime strategy on PC platform”. Application research of computers. 2009,26(9):3575-3577

[5] Liu XiKui, “Design and realization of really 3D visualization system for real-time rendering of large scale terrain,” Application reseach of computer 2009,26(9):3575-3577

[6] Tanner,C.MIGDAL,AND JONES, M. 1998. The clipmap :A virtual mipmap. ACM SIGGRAPH 1998,151-158.

[7] Dave H.Evberly. 3D Game Engine Design. POSTS&TELECOM press 2008.p113-114.