Rendering of Large Textures for Real-Time Visualization

Gabriel Costa Backes* Alex Frasson Tiago Augusto Engel Cesar Tadeu Pozzer

Universidade Federal de Santa Maria, Brazil

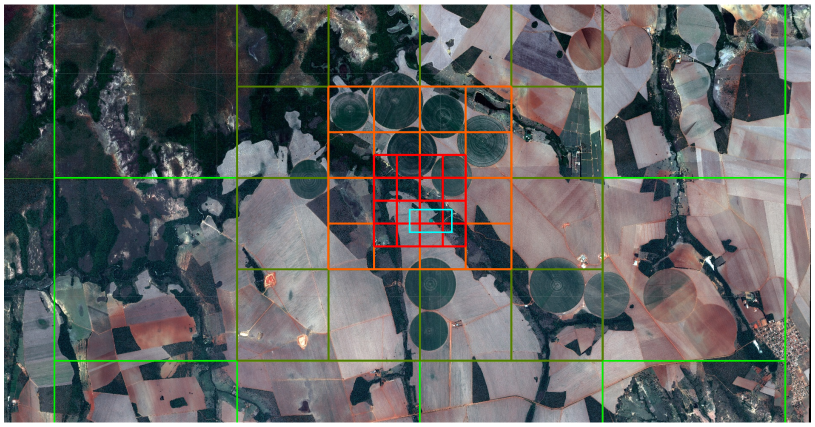

Figure 1: 8 GigaPixel satellite imagery texture rendered at real-time. Each clip region level and its tiles set are represented with colored squares. The cyan rectangle represents the field of view (FOV).

Abstract

Large textures that do not fit in system memory are required to be stored on disk and loaded on demand at runtime. This process may cause performance bottlenecks, compromising interactivity on applications such as GIS, virtual simulations and games. We present a new technique for efficient texture mapping in real-time applications. Our technique preprocess the original texture and saves it as a tiled mipmap texture. At run time, the preprocessed texture is mapped to a quadtree structure and the tiles are loaded on demand, maintaining the real-time performance. Inspired by Clipmap [9], we present a solution to select and cache the tiles according to the view position, zoom level and screen resolution in order to decrease the disk reading request rate. As a validation test we have rendered at real-time an 8 GigaPixel satellite imagery texture on a 84 inches touch screen table that allows the zooming and panning of the texture.

Keywords: Real-Time Visualization, Large Texture Rendering, Clipmap

1 Introduction

Texture mapping is used to add details to a surface in graphical applications. For real-time maps visualization, detailing the region is essential to allow the user to understand the environment and interact with the application. Often, applications such as GIS and virtual simulations use satellite images with very high resolution, making it impossible to store the entire texture in the system memory. Therefore, it is necessary to cache portions of the texture at a time. To facilitate texture manipulation, many approaches subdivide the entire texture into a set of uniform tiles and decreasing resolutions [3][5][7]. Hence, the problem is focused in an efficient paging strategy to maintain the efficient rendering of the subset of most relevant tiles.

*e-mail: gbackes@inf.ufsm.br

In this paper, we propose a new technique for interactive display of very large textures that can adapt to different resolutions. We do not handle elevation maps and other vector data commonly found on GIS applications. Therefore, tiles of the texture are represented with a single quad primitive and the scene is rendering with orthographic projection. Our caching algorithm is inspired by the clipmap approach [9]. The relevant texture tiles are selected by clip regions, which are related to the current view position. If the tiles are not found in the cache, they are first paged asynchronously into system memory. When a clip region is entirely loaded, only the tiles that intersect the field of view are sent to video memory for rendering.

Our texture rendering technique achieves the following goals:

- Any texture size should be supported, non-power-of-2 textures should be managed efficiently;
- The system should work on any resolution;
- Allows user interaction at any time, even while loading the texture tiles;
- The frame rate should remain stable during the loading of texture tiles.

2 Previous Work

Several approaches have been proposed to handle the visualization and storage of large textures. Many of them subdivide the texture into tiles and arrange them in a pyramidal structure [3]. Cline [2] used a quadtree to represent a mipmap hierarchy and presented an approach for texture caching to ensure interactive frame rates during loading steps, even with a limited bandwidth. Hua et al. [5] introduced a similar approach, using a multiresolution model, called texture mipmap quadtree, and incorporated it to a level of detail system.

Tanner et al. [9] presented the clipmap, which supports an arbitrarily large texture while rendering at real-time rates. The main concept is to cache a subset of a mipmap pyramid in the video memory. However, it was developed for a specific graphic hardware. Soane et al. [8] proposed a similar implementation focused on personal computers. Li et al. [7] implemented the clipmap using programmable shaders on GPU.

None of the approaches above use an orthographic view for the visualization, since they are used in terrain rendering, where texture LOD is related to distance from geometry to camera. The further away from the camera, the less detail is required to accurately represent the geometry, since fewer pixels will be occupied by the geometry on the screen space. Although the above approaches work for flat surfaces, they are optimized for perspective projection.

3 Proposed Solution

Our technique has two steps as shown in the Figure 2. The first one is a preprocess step that converts the original texture into another one that will serve as input to our application. We call this resulting texture as *virtual texture*. The virtual texture is formed by the original image divided into uniform blocks at successive power of two dimensions. The color pixels format used was RGB565. No compression scheme was used. After created, it is stored on disk. This step is presented in detail in the section 4.

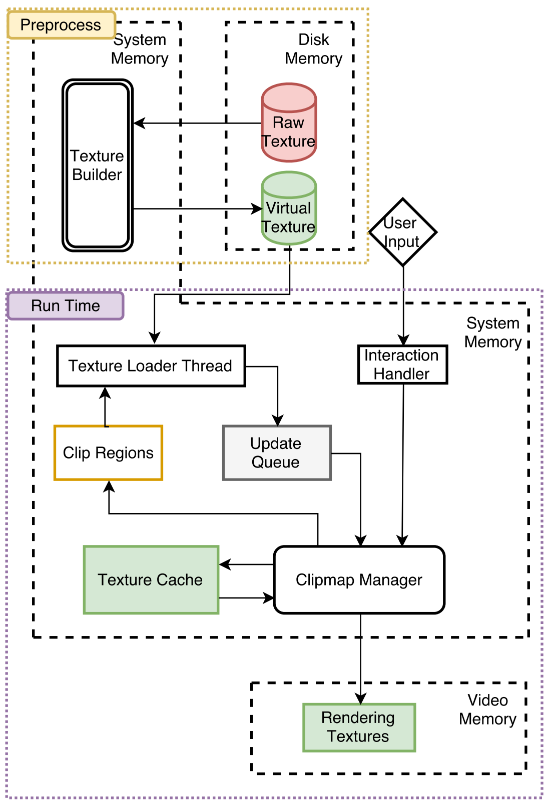

Figure 2: Data flow. The virtual texture is created in the offline preprocessing step. At run time the management and cache of the virtual texture subset is made by the clipmap manager based on the clip regions and the FOV.

The second one, at runtime, caches a subset of the virtual texture. To compute this subset, we implemented data structures that form the *clipmap*. The clipmap consists of clip regions set (Figure 1), which in turn are constituted of a quadtree nodes set. These structures are deepened in the section 5. When a clip region is updated, its nodes are then processed and the required blocks are fetched from disk. The virtual texture blocks loading is made asynchronously and is inserted to an update queue once completed. The node's status is updated when its relative block is dequeued by the clipmap manager. This process is detailed in the section 6. To optimize the rendering, loaded nodes are cropped by the FOV before being sent to video memory. This process is discussed in the section 7.

Implementation and tests were made using the engine Unity3D. However, our approach is generic and can be implemented on any graphical API or integrated into any modern game engine.

4 Creating a Virtual Texture

The virtual texture is created in an offline preprocess step. It uses as input the original texture, called *raw texture*. The raw texture is subdivided into uniform blocks of 2^n x 2^n pixels, and resampled recursively forming a mipmap chain, that at run time will be mapped to a quadtree structure. Each tile corresponds to a quadtree node and is addressed by the coordinates (x, y, l), representing column, row and level, respectively. Each quadtree node represents a quarter of it's parent area while maintaining the same resolution, effectively increasing detail. The root node contains the coarsest texture resolution while the leafs contain full resolution tiles.

For each mipmap level, paddings are added to allow the raw texture to be of any size. The paddings make the final resolution multiple of the node size of the current level. As a result, quadtree may occur to be incomplete, since empty tiles are not saved (Figure 3). The final dimension *d*, in pixels, of the mipmap level can be found by the following equation:

$$d = Ceil(\frac{r}{t}) * t \tag{1}$$

where *r* is the resolution of the raw texture and *t* is the size of the node at current level.

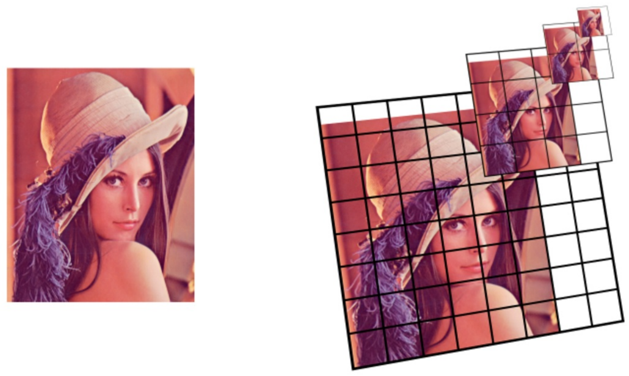

Figure 3: Representation of the tiled mipmap texture. Successives levels have half resolution. The smallest amount of tiles is used to represent each mipmap level. Therefore, empty nodes are not stored.

Although the extra padding makes the virtual texture bigger, taking up more disk space, working with uniform tiles in power of two gives us some advantages. It facilitates the manipulation in order to load and select the tiles. The loading time becomes consistent, since the tiles will be the same size. The virtual texture file size is given by the following equation:

$$S = \sum_{i=1}^{m} (\frac{d_i}{2^i})^2 * bpp \tag{2}$$

where *bpp* is the color depth in bytes and *m* is the mipmap level count.

5 Node Selection

Since only a portion of the texture will be rendered, we need to determine which nodes are present in the FOV and which quadtree level is appropriate. Clip regions are created for each quadtree level. To ensure that the blocks to be loaded cover the full screen space, the clip region dimension n, in nodes, is given by the following formula:

$$n = Ceil(\frac{Max(sh, sw)}{t}) * 2 \tag{3}$$

where sh and sw are, respectively, the screen height and width.

The clip region is centered on c, obtained by the formula:

$$c = Floor(\frac{f}{t}) \tag{4}$$

where f is the FOV center.

An effective region is defined as the area centralized in the clip region center of size *floor(n / 2)*. During the viewer motion, while the viewer position remains inside the effective area, no other tile loading is required. Once outside the effective area, the clip region center is updated and new tiles of the virtual texture are then loaded. This process will be discussed in the following section.

The selection of the most appropriated quadtree level for rendering can be understood as the explicitly compute of the mipmap. When a single texel is mapped to more than one pixel, causing a texture stretch, we need to compute a magnification, in other words, descend on quadtree. In opposition, when more than one texel is mapped to a single pixel would cause aliasing, so we need to ascend on the quadtree to compute a minification.

To simplify this process, we compute the relation between the camera orthographic size, which represents the viewport half height in world space units, and the dimension of each quadtree node. Therefore, this relation λ is given by the following formula:

$$\lambda = \frac{Clamp(Floor(\log_2(o * 2)) - \log_2(u), 0, m)}{\log_2(ns)} \tag{5}$$

where λ values closer to 1 represent more appropriate level. l is the mipmap level, o is the orthographic size, u is the uniform block size and ns is the node size.

6 Update

Once the clip region center is updated, it is necessary to compute the new nodes set that compose the new clip region (Figure 4). Uncached nodes are then fetched from disk and nodes that are no longer present in the clip region can be unloaded. However, remaining nodes from the old clip region do not need to be reloaded.

When a clip region is updated, a list of current region tiles is created. Then, for each tile in this list, the quadtree is accessed by the coordinates of the respective tile and by the level of its clip region. There are three possibilities:

1. The node is present in the quadtree and is already loaded;
2. The node is present in the quadtree and is not loaded;
3. The node is not contained in the quadtree (invalid), that is, the FOV is out of texture bounds.

The size of a clip region is calculated so as to cover all screen space for appropriate quadtree levels. Therefore, as previous levels covers a larger area, we do not need to load all levels fully. The most efficient way is to start loading at the closer level which a single tile covers the current clip region entirely. The tiles present in the levels between it and the current level are just apt to be loaded if intersects the current clip region. The size relation s among two levels i and j is given by:

$$s_i = s_j * 2^{i-j} \tag{6}$$

The start loading level l is given by

$$l = nl - \log_2(n) \tag{7}$$

where nl is the current node level.

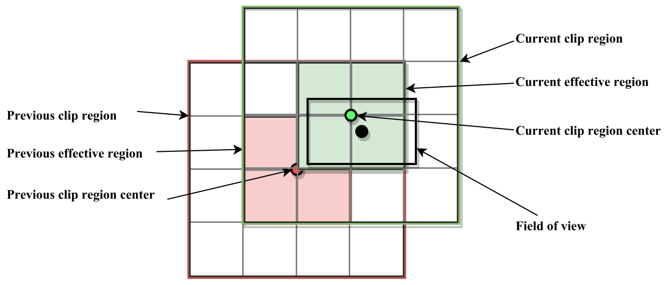
Current clip region
Current effective region
Previous clip region
Current clip region center
Previous effective region
Previous clip region center
Field of view

Figure 4: Clip region update. When the FOV center moves out the effective region, a new nodes set is cached in the system memory. Nodes present in both clip regions do not need to be reloaded.

A resulting list containing quadtree nodes that are not yet loaded is then generated and added to the loading queue. The loading process of a clip region may be slow, locking the application during this process. To avoid this issue, we use threads for loading each virtual texture tile in parallel, guaranteeing the interactivity even if the entire clip region has not been loaded yet.

As soon as a tile loading is finished, its correspondent node is then inserted into an update queue, which contains a node stacks for each clip region. Every frame, the clipmap manager checks the update queue from coarser levels to the current mipmap level. To guarantee a quick progressive detailing, a level starts to be updated only if its parent has been loaded.

After removing a node from the stack, the clipmap manager verifies if the node's tile is still present in the list of the current region tiles. If it is not, it means that the view position has moved, the clip region has been updated and the tile should be unloaded. Otherwise, the quadtree node is marked as loaded.

When the application starts, the coarsest mipmap level is loaded and kept in memory during the execution. Since this level is made of an unique tile, its memory consumption does not affect the application efficiency. In this way, the texture coverage area is always displayed, even before the rendering of detailed levels.

7 Rendering

With the clip region loaded, only the nodes that lie in the FOV, called active nodes, will be sent to the video memory. The loaded nodes are not sent at once, but for several frames, ensuring a consistent framerate. The remaining nodes are kept in system memory for FOV update reasons, decreasing the disk access rate whereas it will be only needed when the camera moves out of the effective region.

After performing the selection of the nodes present in the FOV, for each of them it is necessary to create a texture from the raw data loaded and map to a geometry in the scene. Then, a texture of uxu size is created. The data is recovered from converting the loaded byte array.

Lastly, we need to compute the world coordinate and draw the geometry where the texture will be mapped. A quad mesh is then created and its vertices, uv mapping and triangles are setted. As the quadtree node is addressed with the (x, y, l) coordinates, its position can be found multiplying the x and y by the node size. The texture scale depends on the current node size, since its size is relative to its level. Hence, the texture may be expanded when mapped to the geometry.

Screen Resolution (pixels)	ClipSize (nodes)	Loading Time from HD (s)	Loading Time from SSD (s)
1280x720 (HD)	16	0.30543	0.20191
1920x1080 (FHD)	16	0.31337	0.20768
3840x2160 (UHD)	64	0.65707	0.64730

Table 1: Loading statistics for different resolutions using the SSD and HardDisk as external storages.

8 Results

We have implemented our approach on a personal computer with the following hardwares: 3.2 GHz AMD Ryzen 5 1600, 16GB DDR RAM, NVIDIA GeForce GTX 1070 with 8GB video memory. We have tested with two differents external storages: a SATA HD and a SSD with 156 MB/s and 500 MB/s transference rate, respectively.

We use an sattelite image covering a region about 130 x 69 KM area. The resolution of it represents about 8 GigaPixel. The uniform block size adopted in the preprocess was 1024 x 1024 pixels with 16 bits color depth per pixel. The virtual texture file uncompressed occupied about 21,9GB in external storage.

The table 1 shows the loading time of an entire clip region in differents resolutions and external storages. The loading time with the SSD was approximately 30% faster than the Hard Disk for the HD and FHD resolutions. The UHD resolution performance remained the same for both external storages, which implies that for large amounts of blocks, the loading process stop being the bottleneck. Therefore, removing the node from the stack and send it to the video memory becomes the slow process.

The test was a visualization of the area in the most detailed resolution with the camera moving about 1KM/s. The external storage used was the SSD with 3 differents screen resolutions.

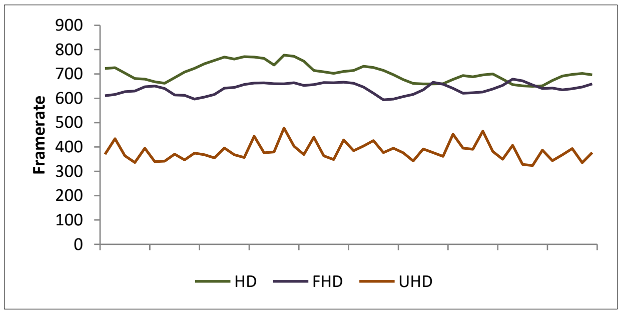

Figure 5: Test results for the following screen resolutions: 1280x720 pixels (HD), 1920x1080 pixels (FHD) and 3840x2160 pixels (UHD).

The figure 5 shows the frame rate for a 10 seconds interval. As expected, the clipsize strongly influences the frame rate. However, even if the clipsize is the same for two different resolution, there may be a slight variation in the frame rate. This is due to the fact that smaller FOVs require less nodes to cover it. Hence, it is necessary to render less tiles.

9 Conclusion

We have introduced a technique to render a large texture while maintaining the real-time performance. Keeping the clipmap essence, we have improved the texture caching for the orthographic view. The virtual texture and the loading of its blocks subsets asynchronously allowed to maintain the interactivity while displaying large textures.

Our implementation does not use any compression algorithm in the preprocess, saving the virtual texture as a raw color array of sequential tiles. There are many researches in this line, which propose to reduce the file size, and thus decreasing the transfer time, while ensure a real-time decompression [1][6][4]. In the future, we expect to explore compression techniques associated to the GPU to improve our approach.

10 Acknowledgements

We thank the Brazilian Army for the financial support through the SIS-ASTROS project, developed in the context of the ASTROS 2020 Strategic Project.

References

[1] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from Compressed Textures. *Siggraph 1996*, page 4, 1996.

[2] D. Cline and P. K. Egbert. Interactive display of very large textures. *Visualization '98. Proceedings DOI - 10.1109/VISUAL.1998.745322*, 98:343–350, 1998.

[3] J. Dollner, K. Baumann, and K. Hinrichs. Texturing techniques for terrain visualization. *Visualization 2000. Proceedings*, pages 227–234, 2000.

[4] S. Guthe, M. Wand, J. Gonser, and W. Strasser. Interactive rendering of large volume data sets. *IEEE Visualization*, D:53–60, 2002.

[5] W. Hua, H. Zhang, Y. Lu, H. Bao, and Q. Peng. Huge texture mapping for real-time visualization of large-scale terrain. page 154, 2004.

[6] Y.-S. Kwon, I.-C. Park, and C.-M. Kyung. Pyramid texture compression and decompression using interpolative vector quantization. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 191–194. IEEE, 2000.

[7] Z. Li, H. Li, A. Zeng, L. Wang, and Y. Wang. Real-time visualization of virtual huge texture. *Proceedings - 2009 International Conference on Digital Image Processing, ICDIP 2009*, pages 132–136, 2009.

[8] A. Seoane, J. Taibo, L. Hernández, R. López, and A. Jaspe. Hardware-independent clipmapping. 2007.

[9] C. C. Tanner, C. J. Migdal, and M. T. Jones. The Clipmap : A Virtual Mipmap. *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer Graphics and Interactive Techniques*, pages 151–158, 1998.