A virtual memory system for real-time visualization of multi-resolution 2D objects

Sergio Pinheiro
Luiz Velho

Departamento de Informática, PUC-Rio
Rua Marquês de São Vicente, 225, Rio de Janeiro, Brasil

IMPA-Instituto de Matemática Pura e Aplicada
Estrada Dona Castorina, 110, Rio de Janeiro,Brasil

Abstract

We describe a predictive memory management system that allows the visualization of 2D graphic objects in real time. This system is based on a virtual memory model. We show how the page loading system predicts and fetches data before it is accessed by the application and how the page replacement system decides which data should be removed from memory. We demonstrate the effectiveness of the system for real-time visualization of large panoramas.

Keywords: texture, virtual memory, cache management, panorama, visual simulation

1 Introduction

Two-dimensional data is widely used in computer graphics. Applications are developed to allow real time rendering of 2D data, for example, visualization of virtual panoramas, images and terrain.

In order to be interactive, those applications should maintain a minimum and constant frame rate. The difficulty of accomplishing this task is related with the amount of data that one wants to visualize. This happens basically due to two factors:

- In the visualization process it is necessary that data is loaded in high-speed memories. In general, these memories have small storage capacity (see the Figure 1).
- To visualize an object it is necessary to perform geometric and illumination calculations. Those calculations are, in general, very complex.

To solve the first problem, memory management techniques are used. These techniques exploit information about the characteristics of graphic objects to determine the data that is kept in memory at a given time.

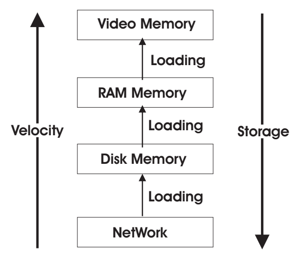

Figure 1: Memory levels: the relationship between storage capacity and bandwidth.

The second problem mentioned above, where is necessary to reduce the amount of processing, can be solved through multi-resolution techniques.

The objective of this work is to develop a memory management system based on the virtual memory paradigm to visualize 2D objects that possess large amount of data. The data is represented as a multi-resolution structure. The system uses predictive loading based on spatial and temporal coherence. In addition, the system also takes into account that the objects are represented in multi-resolution and each resolution level is subdivided in a regular form. That subdivision will allow loading each subset of the data when necessary.

2 Previous Approaches

One of the first visualization systems that used textures in multi-resolution was described in [Willi83]. That system introduced a data structure called mipmap to represent a texture in multi-resolution.

The visualization system presented in [Chist98], uses a representation called clipmap. The clipmap is based on the mipmap, but allows large resolution textures to be visualized in real time. The clipmap employs a system that determines which area of the mipmap should be loaded to texture memory. The main disadvantage of this approach is that it needs a special video card hardware that supports clipmap.

In [Matos98b] a visualization system for virtual panoramas is presented. In that system, the panoramic image is represented at a single resolution level. The memory management mechanism used in this system is based on the paging technique to transfer data among several storage levels.

The system described in [Matos98a], employs panoramic images in multi-resolution. In this work, the memory management system developed in [Matos98b] is adapted to manage panoramic images in multi-resolution. The management system only allows visualization of tiles that are at the same resolution level.

Our work proposes a system for visualization of 2D graphic objects in real time. That system is implemented using off-the shelf hardware. The main requirement is a graphics card with texture capability.

The visualization system is integrated with a memory management system that manages several storage devices during rendering. The memory management suports 2D graphic objects with large textures. The system adapts rendering according to the amount of available memory in the storage devices. That adaptation is based on the multi-resolution structure of the 2D graphic object.

The memory management system can be used in several types of visualization applications, for example, visualization of images, terrain data and virtual panoramas. The system is designed such that communication and data transfer among storage levels are independent of the graphic object type.

3 2D Graphics Object

There are different representations for graphics objects (see [Gomes98]). This work combines two techniques: decomposition and multi-resolution. In the representation process, the graphics object is decomposed in partitions represented at several resolution levels.

The geometric support $U = \bigcup_\lambda U_\lambda$ of the object is decomposed, such that the sets U_i are disjoint. It is also necessary to obtain the attribute $f_{i,j}$ for each element $U_{i,j}$. The attribute function f is a texture. The function $f_{i,j}$ is defined naturally as $f_{i,j} = f \mid U_{i,j}$. Here each $\mathcal{O}(U_{i,j}, f_{i,j})$ is called a *tile*. Observe that the attribute function $f_{i,j}$ associates a subset of texture to the geometric support $U_{i,j}$. This subset can be represented by a subimage, which is a partition of the image that represents the whole texture. In this way, a tile $T_{i,j}$ is defined as $T_{i,j} = (U_{i,j}, I_{i,j})$, where $U_{i,j}$ represents geometric information and $I_{i,j}$ represents texture information.

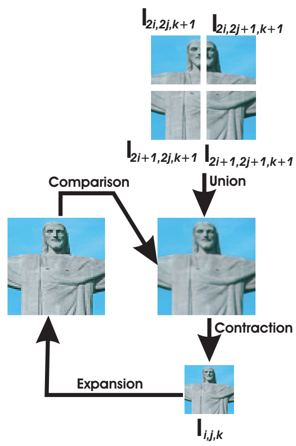

Figure 2: Construction method of the multi-resolution structure.

The graphic object is represented by a matrix of tiles. This matrix of tiles is than converted to multi-resolution. To facilitate the construction of the multi-resolution structure we assume that the dimensions of the matrix are $(2^n \times 2^m)$. Each tile is identified as $T_{i,j,k}$, where (i, j) represents the position in the matrix and k is the resolution level. A tile $T_{i,j,k}$ is defined as union of the tiles $T_{i,j,k+1}$, $T_{i+1,j,k+1}$, $T_{i,j+1,k+1}$, $T_{i+1,j+1,k+1}$

(see Figure 2). Figure 3(a) shows the multi-resolution structure. Observe that each level $k-t$ is represented by a tile matrix of dimensions $(2^{n-t} \times 2^{m-t})$.

In order to reduce the space occupied by the multi-resolution structure, only tiles that increase visual information will possess textures associated with them (see Figure 2). After the simplification process, the multi-resolution structure is shown in Figure 3(b). This representation method is called adaptive multi-resolution.

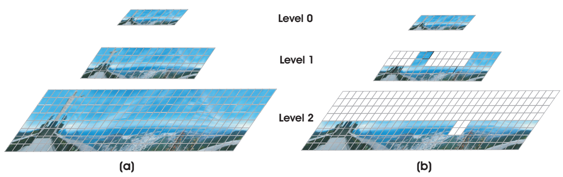

Figure 3: (a)Normal multi-resolution method. (b)Adaptive multi-resolution method.

Table 1 compares the amount of space necessary to store the representation using normal multi-resolution and adative multi-resolution. Observe that the sizes of the textures decrease when represented using the second method. That happens because the adaptive multi-resolution works as a lossy compression.

Size	Levels	Normal	Adaptive
384MB	7	512MB	66.2MB
192MB	6	256MB	63MB
48MB	5	63.9MB	28.5MB
12MB	4	15.9MB	6.93MB

Table 1: Space necessary to store multi-resolution representation.

Each tile is described by a structure that contains four fields: geometric information, texture information, children tiles, father tile. The multi-resolution data structure is a one-dimensional vector, where each element of this vector contains a two-dimensional vector that represents the matrix of tiles at one resolution (see Figure 4). Observe that the lowest resolution level is a forest of quad-trees. Thus, the data structure allows a tile to be located throught the index of the quad-tree structure.

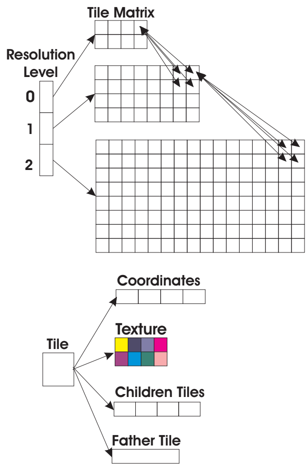

Figure 4: Multi-resolution data strutcture.

4 Memory Management

The memory management system for the visualization of graphic objects was inspired in the virtual memory model [Tanen87][Silbe00]. This model was modified to adapt to visualization problems.

The first modification extends the virtual memory model to manage other types of memory devices. This was made through a generalization of the concept of primary and secondary memory. In the visualization system, a primary memory is a storage device type that has faster transfer rate and smaller storage capacity than the storage device with which is exchanging information. The slowest device with larger storage capacity is considered as the secondary memory. Figure 5 shows the relationship among storage devices.

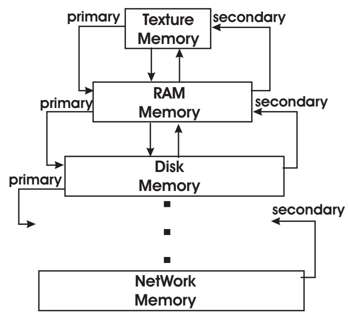

Figure 5: The hierarchy of the storage devices (the concept of primary and secondary device).

The virtual memory system is illustrated in Figure 7. In that system, each storage level possesses a page table used with the same objective that in the conventional virtual memory system, that is, to store the mappings between logical addresses to physical addresses and the state of the mappings. The last storage level doesn't possess a page table because this level is capable of storing all the data. The physical memory is divided into blocks of same size and the logical memory is divided into pages with the same size of these blocks.

The block size, and consequently the page size, depends on the representation of the graphic object. As was discussed in the Section 3, the graphic object is represented by tiles, where each tile is defined as $T_{i,j,k} = (U_{i,j,k}, I_{i,j,k})$. Thus, the size of each memory block is enough to store four images. The decision of maintaining four textures in each block comes from the fact that in the multi-resolution structure, a tile possesses four children. Each tile $T_{i,j,k}$ is represented by a logical address of 4 bytes, expressed in the following way: 2 bits to represent the position of the texture in the page, 22 bits to identify the page, and 8 bits to identify the 2D graphic object. Figure 6 shows a logical address.

Object	Page	Position
8bits	22bits	2bits

Figure 6: A logical address identifies a tile of the 2D graphic object.

The real time visualization system must use a paging mechanism capable to load pages of textures before they are accessed by the application. This mechanism is called *predictive paging*. When a memory management system uses predictive paging, this system is called *predictive management system*.

In a predictive system, page loading and page replacement algoritms are more complex, because they take into account the nature of data and how the data is accessed by the graphic object. Figure 7 shows the predictive paging mechanism.

Observe that in the predictive system is only necessary to have a single page loading module that operates at the highest storage level. When the system makes a request to load a page, this request is passed through several storage levels, in cascade, until the requested page is found. Once the page has been found, it is fetched until becoming accessible at the wanted level. The page replacement system, however, is specific to each storage level. This is because page replacement depends exclusively on the capacity of the storage

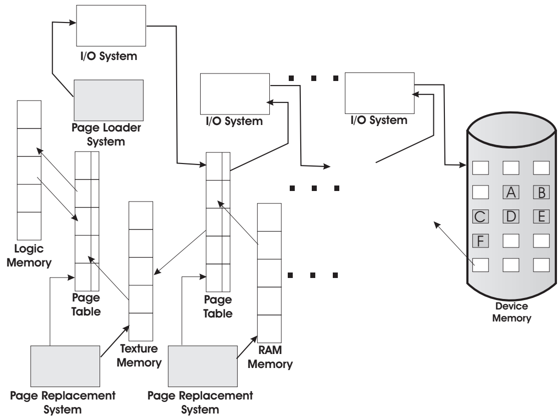

Figure 7: The predictive paging mechanism structure.

device. Thus, for each storage level it is possible to develop a diferent page replacement policy.

The predictive system is implemented as a layer above the operating system. In this work the predictive system was implemented only for texture management. Thus, the virtual address associated to each tile is used just to access its texture. The texture access operations are supplied by the predictive system while the geometry access operations are supplied by the operating system. Figure 8 shows how application communicates with these two memory systems to visualize graphics objects.

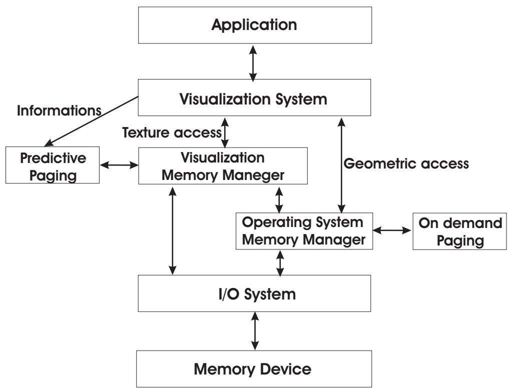

Figure 8: Organization of the memory management system for the visualization of 2D graphic objecs.

The memory size of each storage level is defined at system initialization. The predictive system allocates storage areas and divides them into blocks. The block size depends on the texture size of the tiles.

During rendering, the visualization system ac-

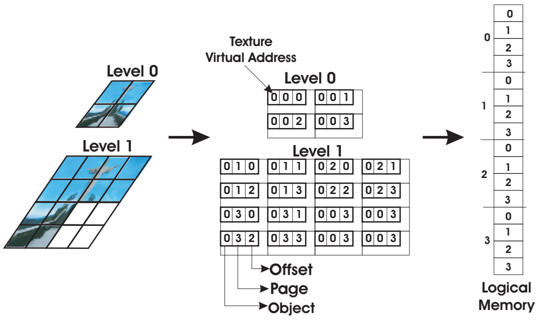

Figure 9: Construction of the logical address space of a 2D graphic object.

cesses the tiles to build a scene. Figure 9 shows how the logical address space of a graphic object is formed.

The memory management system transforms virtual addresses, accessed by the visualization system, in real memory addresses where textures are stored. The predictive paging system will always guarantee that when a virtual address is accessed, it is already mapped to a real address. For that, the predictive paging system will exploit the data structure of the graphic object. Figure 10 shows how the visualization system exchanges information with the management system.

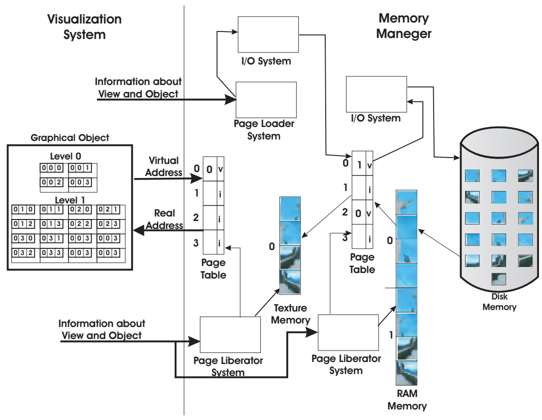

Figure 10: Communication among the visualization and the memory management systems during rendering. This figure illustrates three storage levels.

4.1 Page Loading and Page Replacement

In the visualization process, the virtual camera determines what is shown. Thus, pages are accessed based on the virtual camera.

Mathematically, the problem of predicting camera parameters, can be expressed in the following way: Let $(t_i)_{i\in\mathbb{Z}}$ be an uniform sequence of instants of time $(t_0, t_1, t_2, ..., t_n)$, where $t_i \in \mathbb{R}^+$, and $(p_i)_{i\in\mathbb{Z}}$ a sequence of parameters $(p_0, p_1, p_2, ..., p_n)$, where $p_i \in \mathbb{R}^n$, and a mapping function f defined as $f(t_i) = p_i$, where $0 \leq i \leq n$. Then, we want to find the value of the function f at the instant t_{n+k} based on previous values (t_n), where $k > 0$.

The page loading system solves this problem using the equations that describe the camera movement. That prediction is accomplished through the following steps:

1. Compute the velocities

$$v_{n-1} = \frac{p_{n-2} - p_{n-1}}{t_{n-2} - t_{n-1}}, v_n = \frac{p_n - p_{n-1}}{t_{n-1} - t_n}$$

   and acceleration

$$a_n = \frac{v_n - v_{n-1}}{t_n - t_{n-1}}$$

   .

2. The function f is defined as

$$f(t_{n+k}) = p_n + v_n t_{n+k} + \frac{a_n t_{n+k}^2}{2} = p_{n+k}$$

   , where $k > 0$.

The page loading policy was implemented based on the following rules:

$L1$. In order to visualize a graphic object it is necessary that the texture of the lower resolution level is loaded. Thus, it is assumed that the storage device should be capable to store at least the lowest resolution level of the texture. Figure 11 illustrates this rule.

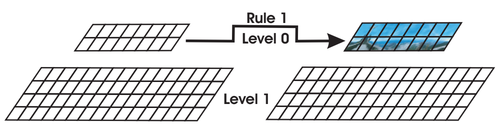

Figure 11: Rule $L1$: The lowest resolution level should be loaded in the memory.

$L2$. If a tile is loaded, then its siblings should also be loaded. This rule is derived naturally because a page stores the textures of the tiles that have a common ancestor. This rule is shown in Figure 12.

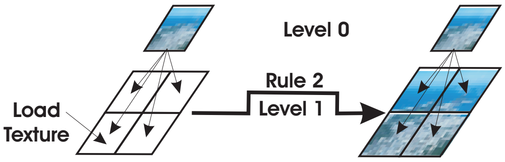

Figure 12: Rule $L2$: If a tile is loaded, then its siblings tiles should also be loaded.

$L3$. If the texture of a tile that is at level k is loaded, then the textures of the ancestrals of this tile should also be loaded. The Figure 13 illustrates this rule.

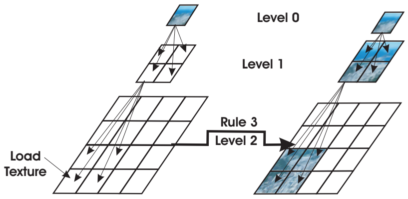

Figure 13: Rule $L3$: If the tile is loaded, then its ancestrals should be loaded.

When a graphic object is visualized, the page loading system, initially, executes four tasks:

1. Load the lowest resolution level of the graphic object (rule $L1$).
2. Calculate the intersection of the view frustum with the geometric support of the graphic object to determine the area that is being visualized. This area determines which tiles are being rendered.
3. Determine the best resolution level that each tile should be visualized. This level is determined based on the projected texture resolution.
4. The pages that contain tiles at the chosen resolution are loaded in the texture memory, obeying the rules $L2$ and $L3$. Thus, when the rendering system draws, these textures are already in the texture memory.

During the visualization process, the rendering system sends the camera parameters to the memory system. Using these parameters, the page loading system predictes the values of the camera parameters for the next k frames. After the prediction phase, the page loading system executes the tasks 2, 3 and 4, for each frame.

In spite of the page loading system effort to load the pages before they are accessed, page prefetching may fail in some cases. This happens due to three reasons:

- The time can be so small that the page loading system can't finish its tasks.
- When the camera moves too fast, it is possible that the system makes a less accurate prediction.
- There is not enough space to store the necessary pages for future frames.

The page loading system cannot interrupt the visualization process to load an unmapped page. This problem is solved exploiting the multi-resolution structure of the graphic object. When an access to an unmapped page happens, the page loading system executes the following task:

- Given that the unmapped page contains the texture of a tile, the page loading system searches the multi-resolution tree to find an ancestral tile that has a loaded texture. Due to rule $L1$, that ancestral can always be found, because all tiles of the lowest resolution level are loaded at initialization. After having found the ancestral tile, the attribute function of the requested tile is modified. Figure 14 shows this task.

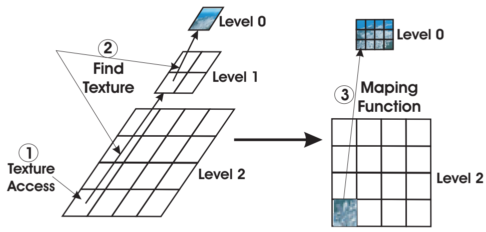

Figure 14: Procedure executed by the page loading system when happens a page fault.

When the page loading system can't load a page due to lack of memory, the page replacement mechanism is activated. Thus, the memory management system calls the page replacement system to decide which page should be removed from memory.

The page replacement system decides to remove a page considering the current state of the camera (position, speed and acceleration). Using this information, it is possible to know which pages in memory are likely to be accessed by the visualization system. Among the candidate pages, the page to be removed is the one at highest resolution level.

After defining the criterion to remove a page, it is necessary also to establish rules with the objective of maintaining consistency between loading and replacement operations. The replacement process is executed obeying the following rules:

$R1$. Pages that contain textures of the lowest resolution cannot be liberated. This rule avoids inconsistency during the execution of the rule $L1$ when page fault occurs.

$R2$. The tile is only liberated after its four children are liberated, Figure 15 illustrates the result of this rule. This rule also avoids inconsistency during the execution of the rule $L1$ when page fault occurs.

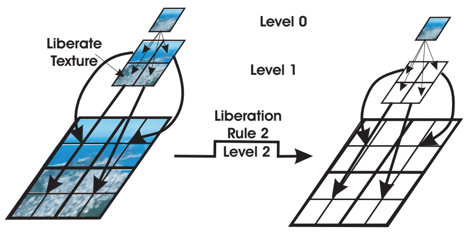

Figure 15: Rule $R2$: The tile is only removed after its four children are removed.

$R3$. Pages that contain textures of the current frame cannot be liberated.

Given that the visualization system is configured to draw q frames per second and that the time spent to render a frame is T_q, where $T_q < \frac{1}{q}$, then the memory management system has time $T_c = \frac{1}{q} - T_q$ to accomplish the tasks of prediction, page loading and page replacement.

5 Application

The memory management system has been tested in a visualization system for virtual panoramas. Figure 16 shows a panoramic image that represents a cylindrical panorama. There are several techniques to create panoramic images (see [Matos98a]).

Initially, to visualize the graphic object the memory management system executes two steps:

1. Load geometric information to main memory. Here, it is assumed that the main memory has the capacity to store all geometric information.
2. Load the necessary textures for the visualization of the current view window. This is accomplished as discussed in the previous section.

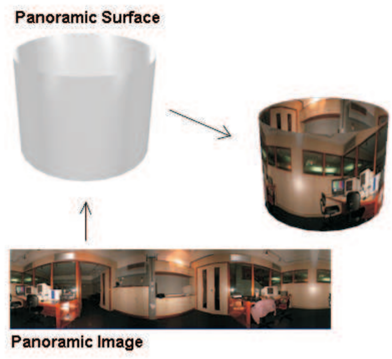

Figure 16: Virtual panorama visualization.

During the rendering process, the page loading and page replacement systems monitor variations of camera parameters (pan, tilt and zoom angles) to predict which tiles should be loaded and which should be liberated. Figure 17 shows the virtual panorama and the texture memory map that indicates which textures are loaded, the current view window and the future views calculated by the prediction system.

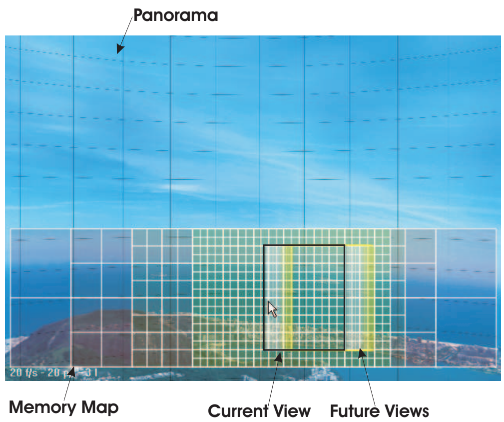

Figure 17: Virtual panorama visualization and the texture memory map.

6 Results

The performance tests of the memory management system were realized on a Pentium III-850

MHz, 786 MB of RAM memory, 40GB SCSI disks with transfer rate of 80Mb/s and Oxygen GVX-420 graphics card. These tests took into consideration the following criteria:

- The size of the panoramic image and the number of resolution levels.
- The amount of RAM and texture memory used during the visualization process.
- The screen resolution has 640×480 pixels.
- The texture of each tile has 64×64 pixels.

In the tests, we used 16MB of texture memory, 128MB of RAM memory and the system was configured to guarantee a frame rate of 30 fps.

Three textures were chosen with the objective of obtaining three different situations. The first texture has 15.9MB and it can be stored completely in both storage devices. The second texture has 48MB and it can be stored completely in the RAM memory, but not in the texture memory and the third texture has 256MB, it cannot be stored completely in neither of the two storage devices.

The application that used only the operating system didn't have any problem to visualize the panorama of 15.9MB. However, during the visualization of the other two panoramas the application didn't reach the minimum rate of 15 frames per second and in some situations the visualization process was stopped for some seconds.

The performance loss was due to two factors. First, the operating system doesn't take into account the nature of the graphic object. Thus, it is unable to exploit coherence of the representation. Second, the operating system uses on demand paging mechanism. In this system the execution is interrupted whenever a page fault occurs.

On the other hand, when the application used the predictive system, a frame rate of 30 fps was easily achieved for the three textures. This happens because the memory management system exploits temporal and spatial coherence during the visualization process. Using that strategy it is possible to develop a paging system that can predict which data will be used in the future. Also, if a tile cannot be loaded on time, the management system searches a texture of lower resolution and gives it to the visualization system. In this way, the execution of the application is never interrupted.

7 Future Works

Various improvements can be done in the memory management system presented in this work. We can mention the following ones:

- Extend the management system to work with geometric data. This implementation will eliminate the limitation that geometric information should fit in main memory.
- Make possible to manage animated textures. This will allow visualization of dinamic environments.
- Extend the system to do memory management during the construction of the graphic object. Thus, large panoramic images can be processed without the need for large storage resources.
- Modify the loading and replacement systems to allow terrain data real time visualization.
- Incorporate the storage network level. This will allow the graphic objects to be stored in server machines that are connected with the client machine runnig the visualization application.

References

[Chist98] T. Chistopher, M. Christopher, and J. Michael. The clipmap: A virtual mipmap. *SIGGRAPH*, pages 151–158, July 1998.

[Gomes98] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and Morphing of Graphical Objects.* Morgan Kaufmann, 1998.

[Matos98a] A. Matos. Visualização de panoramas virtuais. Master's thesis, PUC-Rio, 1998.

[Matos98b] A. Matos, J. Gomes, and L. Velho. Cache management for real time visualization of 2d data sets. *SIBGRAPI*, pages 111–118, 1998.

[Silbe00] A. Silberschatz and P. Galvin. *Sistemas Operacionais.* Prentice Hall, 2000.

[Tanen87] A. Tanenbaum. *Operating Systems: Design and Implementation.* Prentice-Hall, 1987.

[Willi83] L. Williams. Pyramidal parametrics. *SIGGRAPH*, 17:1–11, July 1983.