ヒューマンインタフェース 79－10
（1998. 8. 21）

高精細静止画像のリアルタイムビューア

澤田 研一＊　越塚 登†　坂村 健‡

＊東京大学大学院理学系研究科　†東京大学大学院人文社会系研究科　‡東京大学総合研究博物館

文化的な価値のある資料を電子化し所蔵を行うデジタルミュージアムなどの環境においては、絵画などの平面資料を原本に限りなく忠実に再現するために高解像度の画像を処理することが要求される。資料は容易に電子化されるべきであり、電子化された資料を閲覧するにあたっては高速な表示処理を行えることが望ましい。本論文では、画像を多段階の解像度で階層的に保持することを基本とした、上述の要求を満たす画像の保持形式を提案し、その形式の画像に対して拡大などの操作をリアルタイムに行うための手法について述べる。また、資料を容易に電子化するために、スキャナなどから得られた画像を高精細表示が必要とされる箇所は高解像度で保持し、そうでない箇所は低解像度で保持するための手法についても述べる。

Digital Microscope: A Realtime Viewer for Huge Images

Kenichi Sawada＊ , Noboru Koshizuka† , and Ken Sakamura‡

＊ Graduate School of Science
† Graduate School of Humanities and Sociology　‡ University Museum
University of Tokyo

In the Digital Museum environment, a computer must process extremely high resolution images that represent plane objects, such as paintings, as precisely as possible. These images should be composed easily of captured data, and must be viewed quickly. On the basis of *multiresolution images* – images that have different resolutions in different places –, this paper proposes an image representation which meets the requirements mentioned above, and shows how these high resolution images can be viewed at any arbitrary resolution with seamless zooming in real-time. This paper also describes a method to store plane objects as captured images, allocating high resolution images where it is needed to view in detail and low resolution images where it is not.

1 Introduction

Recently, a number of projects to digitize real objects have been in progress, such as Berkeley Digital Library Project[1], the Digital Museum project[2], and so forth. There are two purposes of digitizing real objects: one for *preservation*, and the other for *exhibition*.

Real objects in this world tend to be weathered. There are a lot of historic paintings, many of which are classified into important cultural property. Although paintings restoration by physical methods is heavily studied, almost all of paintings in the world have been weathered little by little. Such plane objects should be stored as captured images, as precisely as possible in order to preserve the quality of these objects.

On the other hand, time and space for viewing the exhibition in a real-world museum is limited: viewers must go there while the museum is open. *Virtual exhibition* can overcome this limitation by transmitting digital exhibition via networks, such as the Internet. In addition to this advantage, visitors of the museum can make use of magical tools powered by digital technology, such as a magnifying glass for digital images.

In this paper, we present a tool that can give extremely high magnification with smooth interfaces. To develop this tool, we contrived a representation for the target images, a viewing algorithm for this representation, and authoring algorithms to compose this representation. Our representation, *Tile Based QuadTree* (TBQT), can reduce the storage cost by pruning leaves of less necessity for the exhibition, and the algorithm we developed can display demanded TBQT images with fine grained operations: realtime zooming and panning. This paper also describes a technique to compose the TBQT structure by compiling the images captured in different resolutions and different places. We named this integrated system "Digital Microscope", in the sense of a high-performance magnifying glass. This system enables museum managers to compose digital images easily of the source objects, and also enables visitors to view the digital images with smooth and interactive operations. The visitors can satisfy their curiosity for the detailed surface of exhibited objects, without any stress of operations.

2 Related work

2.1 Multiresolution

A conventional method to preserve a plane material is to digitize it into a uniresolution bitmap image. However, this method is inconvenient for interactive uses such as viewing, printing, and editing because the method requires huge amounts of memory and much computing power to handle the images. To overcome this difficulty, many multiresolution image representations have previously been proposed, which can be classified into quadtrees and image-pyramids[13].

Berman *et al.*[5] describes a method to edit multiresolution images by using a quadtree based scheme. It uses a Haar wavelet decomposition and so that it reduces the storage cost to three fourth of the cost using ordinary quadtrees. Although simple quadtree representations have some advantages, this representation must bear the burden of handling pointers. To handle pointers is costly for both computing resources and storage resources.

FlashPix[7, 8] and GRIDPIX[3] are representative image-pyramid formats. They contain multiple images with different resolutions and each resolution image is divided into fixed-size tiles. Users of these formats can easily extract a portion of one image with a proper resolution.

2.2 Zooming interface/mechanism

Although there are several discussions on multiresolution data formats, few viewing algorithms in realtime have been discussed. Berman *et al.*[5] mentions neither any control method for disk storage, nor any real-time operation. Finkelstein *et al.* [9] presents a motion image representation which supports the multiresolution feature by using ordinary quadtrees. The paper mentions real-time displaying for the video application. A unit of disk accessing, however, seems to be one quadtree, so that the method described in the paper is not suitable for our objective — to handle one extremely huge image.

Furnas and Bederson[10] give some detailed analyses for multiscale interfaces. The implementation of Pad[11] and Pad++[4] have a quite good zooming interface. However, these implementations are for structured segmented graphics data such as boxes, lines and so on. We developed smooth zooming interface for large bitmap images as described later.

2.3 Registration

On compiling captured images with different ranges and resolutions into one multiresolution image representation, the compiling algorithm requires the technique called *image registration*, i.e., to detect the overlapping region of two adjacent images. Apple Computer's QuickTime VR[6] presents

a method to make virtual environments with image-based rendering. One of the algorithms described in the paper is *stitching* images. In order to create a seamless panoramic image from a set of overlapping pictures, the *stitcher* uses a correlation-based image registration algorithm to match and blend adjacent pictures.

In our implementation, we chose a sequential registration algorithm described in the textbook by Pratt[12]. Because the algorithm is easy to implement, needs less amount of computation than correlation based registration algorithms, and can give accurate registration enough to compose our image representation.

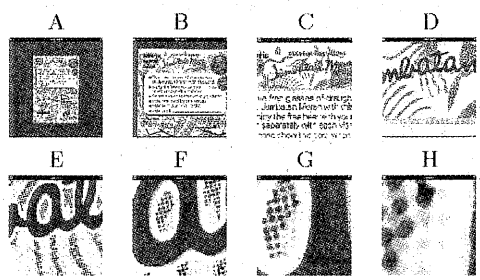

Figure 2: Example of zooming A→H

3 Digital Microscope

3.1 Functional overview

We are now developing the Digital Museum[2]. Visitors of this museum can walk around a virtual space created by the MultiMedia Multi User Dungeon, or *MMMUD* system [16, 14], and appreciate the exhibited objects as they pleases. The visitors chooses an exhibited object, and it is displayed on their terminals with much supplemented information. For example, a portrait of a Japanese famous statesman, *Shôtoku Taishi*, is served with some information about him, such as what he did. In addition to this feature, the visitor can inspect the detailed surface of the portrait by using our Digital Microscope(Figure 1). The user of this tool

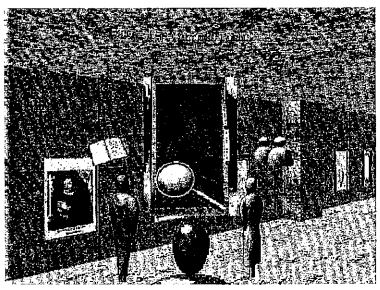

Figure 1: Shôtokutaishi in MMMUD

chooses a point on the image, and then the image is zoomed in. In any level of magnification, the user can drag, or zoom out the image. These operation is performed continuously, without any interruption caused by, for example, data transfer.

Figure 2 shows an example of our implementation. Images in this figure are ordered in time sequences (zooming in). The source object of these figures is a coupon for beers, which is available at one of our favorite restaurants.

3.2 Architectural overview

Digital Microscope contains two runtime modules: Data server and Browsing client (Figure 3). Data

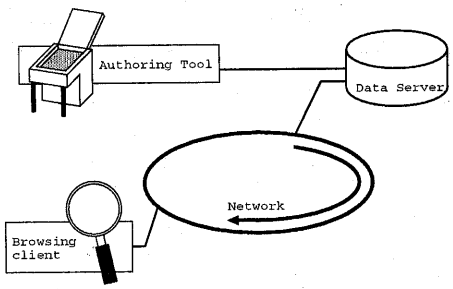

Figure 3: Architectural overview

server preserves huge amounts of exhibited images in a structured format (given in Section 4), and Browsing client provides user interface to view the images. Both are connected with computer networks. Our system currently assumes a relatively fast network such as Ethernet for smooth zooming and panning.

4 Tile based quadtree

We named our image representation "Tile based quadtree", which can be understood as a certain mixture of quadtrees and image-pyramids. This structure is stored in the server side, and a part of the structure is transmitted to the client on demand. This section briefly describes our image representation with some topics such as compression,

and also describes the algorithm to compose this representation.

4.1 Image representation

The goals of our image representation are (1) to enable smooth and interactive viewing operations: zooming in, panning across, and zooming out, and (2) to keep a small storage size: support pruning branches and compressing tiles.

To accomplish these requirements, we propose an image representation scheme. Here, we will give a formal definition for the scheme. First, let s be a integer value, and t be a half of s. Let $P(x, y)$ be a squared matrix of pixel values, whose size is s^2. More formally, P is defined as a mapping $V \times V \mapsto A$ where V is the coordinates value and A is the value that stands for a color of the pixel. In this paper, P is called a "tile".

For *parent node* $P_{m,n,k}(x, y)(0 \leq m, n < 2^k$ of integer, m and n stand for the coordinates value and k does for the depth level, see Figure 4), its children are defined as $P_{2m,2n,k+1}(x, y)$, $P_{2m+1,2n,k+1}(x, y)$, $P_{2m,2n+1,k+1}(x, y)$, and $P_{2m+1,2n+1,k+1}(x, y)$. The parent node can have 0,1,2,3, or 4 children. Each node in this quadtree must satisfy this constraint:

For $0 \leq x, y < s$

$$\text{Let } i = \left\lfloor \frac{x}{t} \right\rfloor, j = \left\lfloor \frac{y}{t} \right\rfloor, u = x \bmod t, v = y \bmod t$$

$$\begin{aligned} P_{m,n,k}(x, y) = \frac{1}{4}\{ & P_{2m+i,2n+j,k+1}(2u, 2v) \\ & + P_{2m+i,2n+j,k+1}(2u+1, 2v) \\ & + P_{2m+i,2n+j,k+1}(2u, 2v+1) \\ & + P_{2m+i,2n+j,k+1}(2u+1, 2v+1)\} \end{aligned}$$

Now, we can define the tile based quadtree as this quadtree with a root node $P_{0,0,0}(x, y)$.

The size of tiles s is changeable corresponding to needed processing speed. In our implementation shown later, the size is 512 × 512 pixels.

4.2 Compression

Because the tile based quadtree scheme bases one tile as a unit, the scheme can use compression techniques for still pictures such as JPEG[15], which cannot be applied to the ordinary quadtree scheme. Although the quadtree, by its nature, can support compression by pruning any branches of the quadtree whose detail is not needed, the compression is not sufficient. Our representation described above can support both of the compression methods mentioned in this subsection.

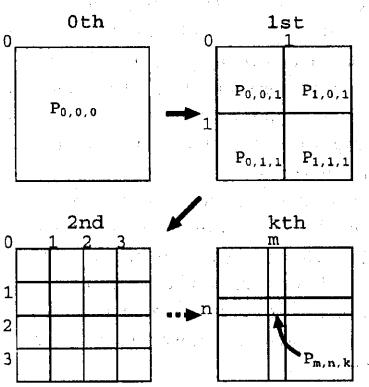

Figure 4: Tile based quadtree

4.3 Automatic data composition

On composing this representation, the user prepares images captured by a scanner device or other capturing devices. The images can have various ranges and resolutions (Currently, the resolution is restricted to a power of 2 of the base image's resolution.) Of course the image with the lowest resolution must range over all of the source plane object. We call the lowest resolution image as "Base image" corresponded with $P_{0,0,0}$. Here, we briefly describe a method to compile these captured images into our multiresolution image representation.

Before main processes, the algorithm chooses a captured image that has the higher resolution than the base image. The order of choosing images is corresponding to the resolution, i.e., the higher resolution the image has, the later turn the image takes. This is because it is more accurate to register the images "near" in resolution — the resolutions the images have are close to each other — than to register the "far" images. The source image the algorithm chose is called "Embedded image". Here, we will show a process to embed the higher resolution image into the lower one.

First, search the portion of the base image that matches best to the embedded image (Global registration). Second, make tiles for the embedded image in the tile based quadtree, and fill them with interpolated images from lower resolution images (Figure 5 left). Third, calculate which position in the interpolated tile has the best matching to embed (Local registration), and embed the image into the interpolated image (Figure 5 right). Last, make all empty nodes between the embedded tile node and the root node to keep consistency in the tile based quadtree, and write back to the disk storage,

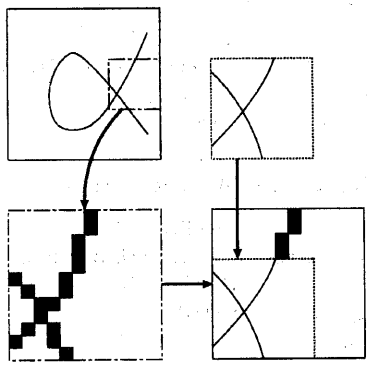

Figure 5: Embedding into interpolated images

and then, repeat these processes until no source images to embed are left. After these processes, we obtain the structure described in Figure 6.

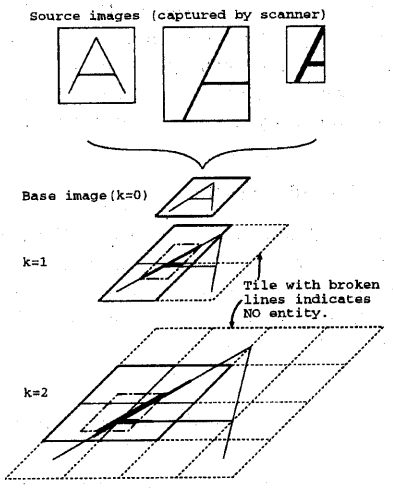

Figure 6: Composed structure

5 Viewing algorithm

Recall that one of the goals of our representation is to display images in real-time, with smooth zooming and panning. To accomplish these goals, we will describe an algorithm to display our multiresolution image structure.

5.1 Image cache block

Recent operating systems use the main memory as a cache device for disk storage, which is called the virtual memory system. Similarly in spirit to this feature, but to cache more smartly, the displaying algorithm prepares *image cache blocks* for the client. Each block can contain one tile defined in the previous section, and each level in the tree has four blocks. Let k be the levels of the tile based quadtree. The algorithm uses $4k$ blocks.

A set of four blocks in each level is called a *quadruple block*. Each block in a quadruple block will have a tile in the corresponding level.

5.2 Window

After creating the quadruple blocks, the algorithm set a window in the block of the lowest resolution (Figure 7). The size of the window is variable and

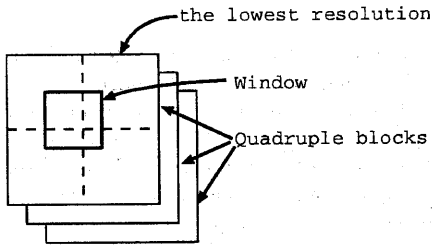

Figure 7: Window

the part of the image included by the window is projected into a fixed square box on the screen (Figure 8). Thus, the window's motion means the so-called scroll operation. When the window shrinks, the image on the screen is zoomed in, and vice versa. If the window size becomes larger or twice smaller than the size of a block, the window jumps to the quadruple block in the next resolution, and thus the size of the window is kept within the range of 0.5 to 1 time as large as that of a block is.

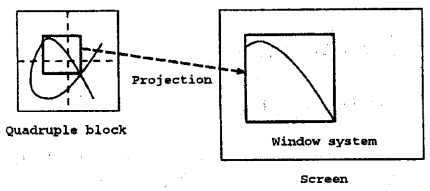

Figure 8: Projection

5.3 Interpolation

When the window described above is moved out of the quadruple block or when the window jumps up or down over resolutions, the quadruple block in the target level may not be ready. In such situations, exact images must be transmitted from the server. The transmission is, however, frequently delayed. To meet the requirement of real-time displaying, the algorithm makes some temporal images by interpolating the lower resolution blocks (Figure 9). This figure illustrates the situation that the window is moved out of the quadruple block. In this situation, there are no exact image data of Area B, so that the algorithm complements Area B by interpolating the corresponding area in the quadruple block of the lower resolution. While displaying the temporal image, the algorithm waits for the transmission to be completed. Area A illustrated in Figure 9 is not useful then and so that Area A is overlaid with the transmitted images. When the transmission is completed, then, the algorithm displays the exact window on the screen. This complement can be executed even the level of available quadruple blocks is two or more levels away from the current level.

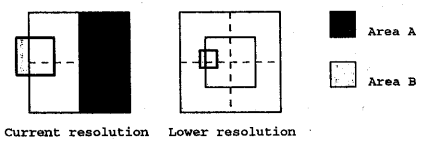

Figure 9: Interpolation

Note that the transmission can be executed only if there are the exact images in the server storage, and otherwise the interpolated image is steadily used. Thus, the algorithm complements pruned branches of the tile based quadtree. The quadruple block with the lowest resolution ranges all over the object, and therefore the complement can be executed wherever the window is moved.

6 Evaluation

6.1 Pilot system

Our system for evaluation is implemented upon the following hardware and software environment. For this evaluation, we have implemented the server client communication upon NFS (Network File System) on Ethernet, to assure enough performance for smooth zooming/panning.

NFS Server: SPARCserver1000

OS	SunOS 5.5
MPU	SuperSPARC 85MHz × 2
Memory	256MB
Hard disk	500GB

Browsing Client: AT compatible

OS	FreeBSD 2.2.5
Window system	X11R6 with Accelerated X
MPU	Pentium Pro 200MHz
Memory	64MB
Network	10baseT

The program we wrote has, about 1000 lines of the *C* language for display and about 600 lines for composition.

6.2 Quadtree implementation

In order to compare the performance of our scheme and the quadtree based scheme [5], we also built another implementation using the quadtree based scheme with a Haar wavelet decomposition. Because it is difficult to transmit the quadtree data partially, the implementation transmits the full of this data, and the client program leaves management of overflowed memory to the virtual memory system handled by the operating system.

6.3 Storage space

We have evaluated the data size of the tile based quadtree format images stored in the server system. As example data with precise details, we have prepared a beer coupon (Figure 2) and a mask pattern of VLSI CPU (TRON CHIP).

Table 1: Sizes (in Mb) of sample images

	Source	QT	TBQT	CTBQT
Beer Coupon	4,551	76	75	8
Tron Chip	192	22	31	4

Table 1 reports our result. The "Source" column reports the total space that would be required to store the all pixels, assuming the entire image had been expanded to its highest spatial resolution present anywhere in the representation. The next column, "QT" gives the total space required to construct the quadtree structure described in [5]. The "TBQT" column reports the total storage size required for our scheme with no compression. The "CTBQT (Compressed TBQT)" column gives

the size required for our scheme with JPEG compression. Of course the storage cost in the latter three formats is reduced by eliminating unnecessary nodes (stored only the detail presented in the example), and this results in the great difference between "Source" column and the others.

6.4 Frames per second

It is a good measure for evaluating smoothness to estimate the value of frames per a second. We measured the time to complete zooming operation presented in Figure 2, A → D and A → H. Since zooming operation requires frequent data transmission and the network is almost saturated, the measured value can be applied to panning operation that also requires much data transmission. Table 2 indicates the value of frames per a second for each scheme.

Table 2: Frames per one second

	Frames	QT		CTBQT	
		Sec	F/Sec	Sec	F/Sec
A→D	56	40.2	1.4	4.47	12.5
A→H	100	n/a		9.0	11.1

The "Frames" row gives a total number of frames in zooming, and the "Time" row indicates the time zooming operation takes in second. The "n/a" entry stands for "not achieved" because of exceeding the virtual memory system's capacity. This table shows that our CTBQT method is about 9 times faster than the QT method. Note that, in this experiment, the zooming rate between nth frame and $(n+1)$th frame is 106.5%. This rate is computed in accordance with a heuristic that the zooming rate should be 200% per a second: $\sqrt[11]{2} \approx 1.065$. Of course this heuristic is at all our discretion.

Theoretically (or, in other words, if our machine has two MPUs: for displaying and for data retrieving), the measured time will depend only on the displaying process. In practice, however, the machine we implemented has only one MPU, and because of this, the result includes the time for transmission, for decompression(JPEG), and for display. Figure 10 gives a breakdown of the total processing time (for A→H).

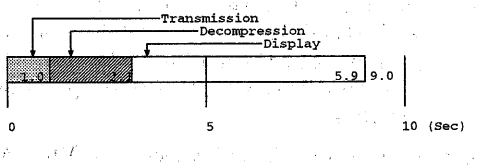

Figure 10: Analysis of the total processing time

7 Conclusion and future work

7.1 Conclusion

There are a lot of methods to handle images with extremely high resolution. In consideration of both storing and computing costs for handling the images, it is reasonable to store the images with some multiresolution feature. To enable people to magnify the exhibited image in realtime, we propose a certain mixture scheme of quadtrees and image-pyramids called *Tile based quadtree*. We also develop a viewing algorithm for this representation, and confirm the smoothness of viewing operation by measuring the value "Frames per second".

Moreover, we develop a system that composes our representation automatically. It is a laborious job to compose multiresolution image structures in any existing multiresolution scheme. To ease the burden of this task, we implemented a program which enables automatic composing of the image structures. The user has only to prepare the image with some resolutions, and the program automatically compile them into our image representation.

7.2 Future work

Although our system already performs a compression by the JPEG scheme in one tile. By making use of the similarity between the parent tile and the tiles of its children, an *inter-tile* compression may be performed. This compression is, however, a tradeoff for the smoothness of viewing operation. It decreases the amount of network communication but increases the computational cost of the viewing algorithm.

References

[1] `http://elib.cs.berkeley.edu/`.

[2] `http://www.um.u-tokyo.ac.jp/digital/`.

[3] Satoshi Asami. GRIDPIX — A Method for Presenting Large Image Files Over the Internet, 1998. available via http as: `http://now.cs.berkeley.edu/Td/GrixPix/`.

[4] Benjamin B. Bederson and James D. Hollan. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In *Proc. of ACM UIST'94*, pages 17–26, 1994.

[5] Deborah F. Berman, Jason T. Bartell, and David H. Salesin. "Multiresolution Painting and Compositing". In *Computer Graphics(Proc. SIGGRAPH'94)*, pages 85–90, 1994.

[6] Shenchang Eric Chen. "QuickTime VR - An Image-Based Approach to Virtual Environment Navigation". In *Computer Graphics(Proc. SIGGRAPH'95)*, pages 29–38, 1995.

[7] Eastman Kodak Company. FlashPix Format and. Architecture White Paper, June 17 1996.

[8] Eastman Kodak Company. FlashPix Format Specification Version 1.0, September 11 1996.

[9] Adam Finkelstein, Charles E. Jacobs, and David H. Salesin. "Multiresolution Video". In *Computer Graphics(Proc. SIGGRAPH'96)*, pages 281–290, 1996.

[10] George W. Furnas and Benjamin B. Bederson. Space-Scale Diagrams: Understanding Multiscale Interfaces. In *Proc. of CHI Human Factors in Computing Systems*, May 1995.

[11] Ken Perlin and David Fox. Pad: An Alternative Approach to the Computer Interface. In *Computer Graphics(Proc. SIGGRAPH'93)*, pages 57–64, 1993.

[12] William K. Pratt. *Digital Image Processing*, chapter 19.5, pages 562–567. Wiley-Interscience, New York, 1978.

[13] S. Tanimoto. *Structured Computer Vision*, chapter 2.3, pages 40–55. Academic, New York, 1980.

[14] Tomonori Usaka, Shunsuke Yura, Kazuhiko Fujimori, Hirohisa Mori, and Ken Sakamura. A multimedia mud system for the digital museum. In *Proceedings of 3rd Asia Pacific Computer Human Interaction*, pages 32–37. Information Processing Society of Japan, IEEE Computer Society, July 1998.

[15] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[16] Shunsuke Yura, Tomonori Usaka, Kazuhiko Fujimori, and Ken Sakamura. Design and implementation of the browser for the multimedia multi-user dungeon of the digital museum. In *Proceedings of 3rd Asia Pacific Computer Human Interaction*, pages 44–49. Information Processing Society of Japan, IEEE Computer Society, July 1998.