Pyramid Texture Compression and Decompression using Interpolative Vector Quantization

Young-Su Kwon, In-Cheol Park and Chong-Min Kyung
Department of Electrical Engineering and Computer Science, KAIST
Kusong-Dong, Yousong-gu, Taejon 305-701, Korea
Tel : +82-42-869-4402
E-mail : {yskwon,icpark,kyung}@vslab.kaist.ac.kr

Abstract—Texture mapping is a common technique used to increase the visual quality of 3D scenes. As texture mapping requires large memory to deal with large textures generally required in the current visual systems, we propose an algorithm for compressing a pyramid texture used for mipmapping. Vector quantization is used to compress all levels of the pyramid texture to one representative value databook, one residual codebook and one index map. The proposed compression scheme uses interpolative texel difference vector quantization that compresses the difference between the interpolated surfaces generated by the representative value and the correct uncompressed texels of the texture at each level. The compressed pyramid texture can be accessed randomly and decompressed without loss of visual quality. We also propose a hardware architecture that performs the trilinear filtering with the compressed pyramid texture.

I. INTRODUCTION

Texture mapping is an effective technique to enhance the realism of a computer-generated object by mapping textures on the surface of the object. A main obstacle of the texture mapping is that it requires large memory to handle large textures required in the current visual systems such as game systems. To render a 3D scene that requires large memory to store texture images, two methods have been proposed, that is, texture compression and texture caching.

In *texture compression*, a large texture is compressed to save it in a memory of small capacity[2][3]. When the texture is needed for texture mapping, a rendering hardware decompresses the compressed texture. Vector quantization (VQ) has been commonly applied to the compression because it allows fast decompression that is a critical factor in real-time rendering. In [2], an image to be encoded is first partitioned into a set of spatially adjacent square blocks. Each texel block is then encoded independently and represented by a vector. A codebook is made based on these vectors and each block is associated with an index that points to a codebook entry. The index is selected such that the code contained in the codebook entry is the closest one to the original texel block. Although this technique eliminates the need to store the large texture in main memory, decompression time is additionally needed in the rendering pipeline.

As another method to deal with large texture, *texture caching* has been proposed to store frequently accessed texture images in a cache memory of small size[4][5]. The texture cache is implemented as a fully associative cache because it is accessed randomly. Although it is effective in reducing the access time if the texture image required is hit in the texture cache, the large original texture must reside in the main memory to cope with the miss case.

Many compression techniques developed for other uses can be applied to texture compression, but VQ was mainly used in the previous works because of two issues related to the texture compression: random access and decoding speed. As it is difficult to know in advance which texels will be accessed, texture compression schemes must provide a fast random access mechanism. The still image compression schemes like JPEG and run length coding have variable code rate and thus the renderer has to decompress a large texture sequentially in order to extract one texel. In VQ, each pixel block is represented by a fixed number of bits to enable the random access. In order to render directly from the compressed texture, decompression should be fast enough such that the time to access a single texel is not severely impacted. Fast decompression is possible in VQ, because decompression can be achieved by table lookups.

We have chosen the VQ texture compression scheme to store the pyramid texture used in mipmapping. Mipmapping is an effective way to resample textures[1]. In mipmapping, a texture is stored as an image pyramid in which each texel of the pyramid level is a filtered version of the corresponding texels of the original lowest level texture. The compression scheme for the pyramid texture is proposed in [2], but each level of the pyramid texture requires its own codebook. Another drawback is that the quality of a higher level codebook can not be better than the quality of a lower level codebook because the codebook of the higher level is obtained by filtering the lower level codebook. To overcome these drawbacks, we propose a new compression technique that is effective for pyramid textures.

This paper is organized as follows. In Section II, the proposed method for the pyramid texture compression is explained and the experimental results are shown in Section III.

II. PROPOSED PYRAMID TEXTURE COMPRESSION

The vector quantization is a process that maps a set of data into a predefined set of patterns represented in the form of vectors. Fig. 1 shows the vector quantization of input image. It is a mapping from a k-dimensional Euclidean space R^k to a finite subset of R^k. The finite set is called the VQ codebook. The encoder generates the index to the codebook that is closest to the current encoded vector. The codebook is generated by LBG algorithm that is also known as the generalized Lloyd algorithm. After the size of the codebook is determined, the initial training vectors that are the complete set of the difference vectors are gained from the partitioned texture block. The initial codebook entries called codewords are selected from the training vector and each training vector is grouped to the nearest codeword. After that, the median in each group is selected as a new codeword. This process is it-

0-7803-6297-7/00/$10.00

erated until the codebook that satisfies the required quality is found.

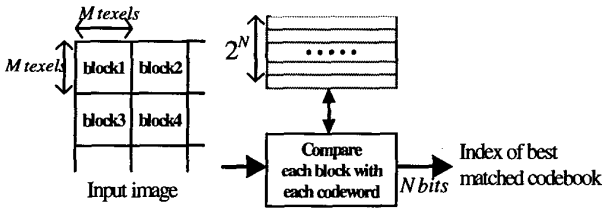


Fig. 1. Vector Quantization.

For a given pyramid texture, the proposed pyramid texture compression generates two books to represent low frequency terms and high frequency terms, respectively. The first book called a representative value databook is used to reconstruct the low-frequency pyramid texture and the second book called residual vector codebook is used to reconstruct the high-frequency texture. To construct the residual vector codebook, Interpolative Vector Quantization (IVQ) is used to find correlation between the blocks of an image by creating interpolative surface from the representative value of the blocks[7]. The residual between the interpolative surface and the original image is then vector quantized.

The first benefit of IVQ is that it can use the correlation between the corresponding texels of successive levels when it is used for pyramid texture compression. In the proposed *pyramid texture compression*, the representative value databook, index map and the residual vector codebook can be shared for all levels of a pyramid texture. The second benefit is that the blocked-fashion coding of IVQ supports random access and fast decompression. As it is difficult to know in advance how a renderer will access a texture, the texture decompression scheme must satisfy a random access to texels. The still image compression schemes like JPEG and run length encoding has variable codes rates and they require a large portion of a texture to be decompressed to extract one texel. In those schemes, fast decompression is difficult because of the time to access a single texel. Finally, the interpolation of the representative vector improves the visual quality of decompressed texture.

A. Encoding of the Representative Value Databook

The pyramid texture is a set of textures in which each level texture is a filtered version of the lower level texture. The structure of the pyramid texture is shown in Fig. 2, where the size of the lowest level texture is H×W and the texture size of level *i* texture is a quarter of the texture of level *i-1*. The representative value databook is made based on the texture of the lowest level, l_0 which is segmented by N × N blocks. The representative value of each block may be the average color or one of the texel color in that block. If the bit width of one color is C, the size of the representative value databook is $(H \times W)C/N^2$. In the proposed pyramid texture compression, only one representative value databook is used because each level texture is reconstructed from the representative value databook as explained in the following section. The structure of the representative value databook is shown in Fig. 3 where P and Q are the maximum column and row numbers in the representative value databook.

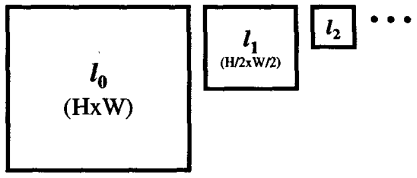


Fig. 2. The pyramid texture.

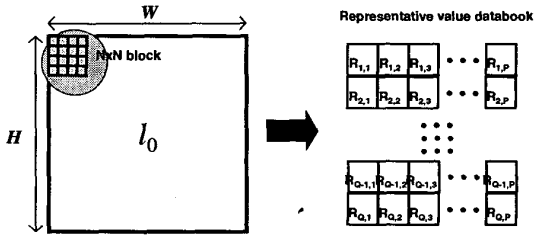


Fig. 3. The representative value databook.

B. Encoding of the Residual Vector Codebook

The low frequency part of each level texture is reconstructed from the representative value databook. The residual vectors are the difference between the original texture and the reconstructed texture. They represent the high frequency part of the original texture. These difference vectors are used as the training vectors to generate the residual vector codebook.

The algorithm to generate the training vectors for the residual vector codebook is shown in Fig. 4. Initially, we set the training vector set TR to ϕ, and for each block $B_{q,p,l}$ on the level l texture, the block $RB_{q,p,l}$ is reconstructed from the representative values. Two candidates for the representative value of (q,p) at level l are shown in the following equation :

$$R[q_l, p_l] = 2^{-2l} \sum_{i=0}^{2^l-1} \sum_{j=0}^{2^l-1} R_{(2^l q_l + i),(2^l p_l + j)} \text{ or } R_{(2^l q_l),(2^l p_l)}.$$

The first one is the average value of all the corresponding representative values of the current block. The second one is the corresponding representative value for the level l texture. The second one is very simple compared to the first one and needs no other computation. In the proposed algorithm, the second one can be used because the residual vector codebook will annotate the high-frequency term. The training vector is the difference vector between $RB_{q,p,l}$ and $B_{q,p,l}$. The training vector generation algorithm is shown below where R[q,p] is the representative vector at (q,p).

```
TR = φ;
For each B_{q,p,l},
  RB_{q,p,l} = Interpolate(
        R[q×2^(l+1), p×2^(l+1)], R[(q-1)×2^(l+1), p×2^(l+1)],
        R[q×2^(l+1), (p-1)×2^(l+1)], R[(q-1)×2^(l+1), (p-1)×2^(l+1)] );
  Residual = Diff(B_{q,p,l}, RB_{q,p,l});
  TR = TR∪Residual;
```

Fig. 4. Algorithm for texture reconstruction.

After the training vector set is generated from each level of the pyramid texture, the LBG codebook generation algorithm is used. First, n initial vectors are generated by the *initial codebook generation method* and then the training vectors are grouped with those initial vectors to make n groups. The me-

dian value is found and registered as a new codebook vector in each group. This procedure is iterated until a codebook that satisfies the required quality is found. In each iteration, the index map to the residual vector codebook can be determined by the *index map determination method.* The index map determination and the initial codebook generation methods are described in the experiments. The process for encoding of the residual vector codebook is shown in Fig. 5.

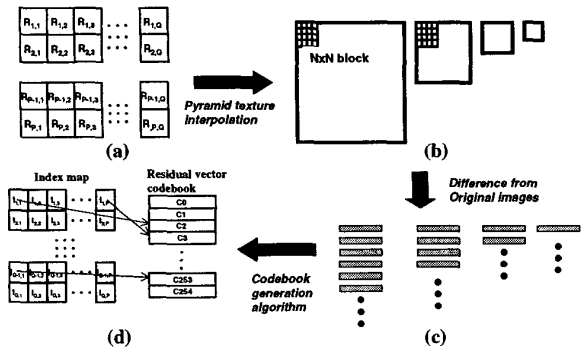


Fig. 5. Encoding of the residual vector codebook. (a) Representative value databook. (b) Interpolated pyramid texture. (c) Index map and residual vector codebook. (d) Training vector set from each level.

C. Decoding

In the traditional 3D rendering architecture, the 3D renderer reads the texel value from the texture memory that contains the whole textures for the current 3D scene. While the texture storage unit of the traditional architecture is composed of large memories, the proposed architecture is composed of small memories and some logic blocks that are used for the computation of the block address, interpolation and residual vector annotation.

A hardware structure of the pyramid texture decompression is shown in Fig. 6. The representative value databook, the index map and the residual vector codebook for the current texture are downloaded from the host processor. When the renderer requests a texel at *(i,j,l)*, the (q, p) coordinates of the representative databook are computed and the four representative data are read from the representative vector databook. They are interpolated to reconstruct the low frequency part of the block. The index to the residual vector codebook is read from the internal memory of the rendering hardware concurrently with the block reconstruction. The high frequency value from the residual vector codebook is annotated to the interpolated block to generate the final texel block.

The decompressed texel blocks are saved in the decompressed block cache. The decompressed block cache not only increases the performance of the renderer but also enables the trilinear filtering. If the renderer is based on the trilinear filtering, it is required to read 8 texel values simultaneously because it averages 8 texel values that are four at level *l* and four at level *l-1* for one pixel in the screen. The decompressed block cache contains previously decompressed blocks for each level and it enables the renderer to read texels simultaneously.

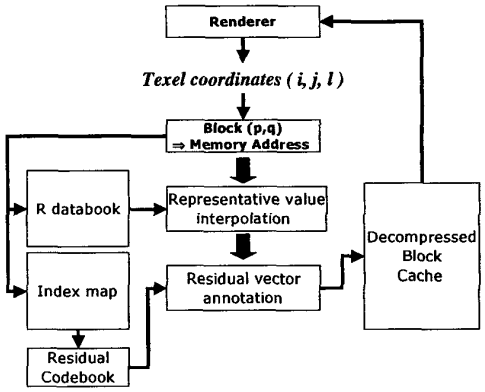
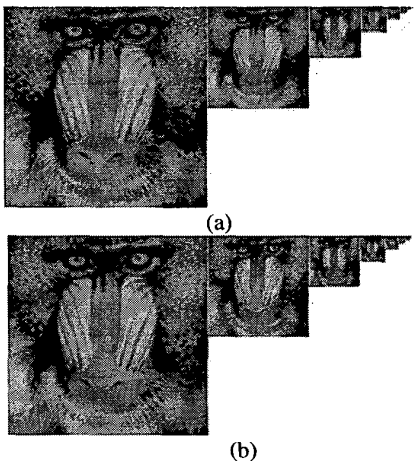


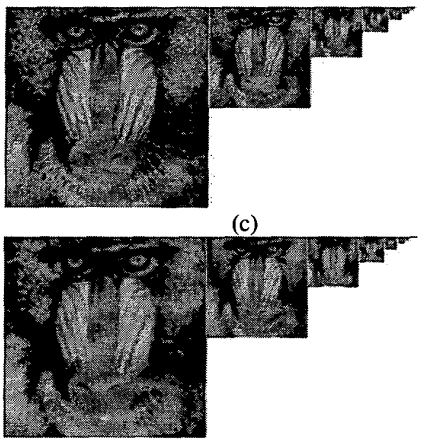
Fig. 6. Pyramid texture decompression hardware.

III. EXPERIMENTS

The proposed pyramid texture compression scheme is applied to a pyramid texture generated from the 512x512 baboon image. The mean value of each block is used as a representative value and the difference vectors for each texture image are extracted. The compression ratio is 10:1 for 4x4 block and 40:1 for 8x8 block. In the LBG algorithm used to generate the residual vector codebook, three methods are tried for index map determination and the initial codebook generation.

The first scheme selects an initial codebook from the lowest l_0 texture and the index map to the codebook is determined by the minimum distance between the l_0 texture and the codebook. The second method uses an initial codebook obtained from the average of the corresponding blocks at each level and the index map is determined based on the closest distance between the l_0 image and the codebook. This method considers the effect of the higher level textures. The third method uses the initial codebook from the average of the corresponding blocks and the index map based on the closest distance between the average of the each level of the pyramid texture and the codebook. Fig. 7 shows the images decompressed using the three index map and codebook generation schemes. The original image is shown in Fig. 7.(a).





(c)

(d)

Fig. 7. Comparison of decompressed pyramid textures. (a) Original pyramid texture. (b) Decompressed pyramid texture using codebook generation scheme 1. (c) Decompressed pyramid texture using codebook generation scheme 2. (d) Decompressed pyramid texture using codebook generation scheme 3.

The images (b) and (c) show the high frequency feature of the original image very well. The pyramid texture in (d) does not show the high-frequency effects because we have chosen the initial codebook and the index map based on the multi-level texel averaging.

Fig. 8. shows the PSNR of the three schemes. The PSNR of the first and the second codebook generation method is almost the same and it is reduced for the higher level. The third method maintains the higher PSNR for the higher levels but the quality of the lowest level is not good.

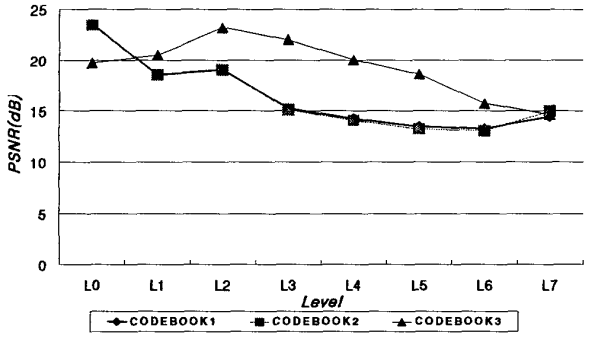
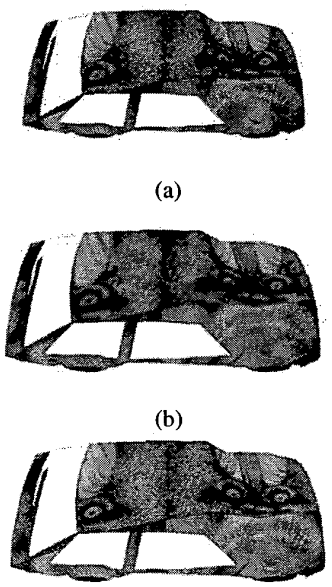


Fig. 8. PSNR of the decompressed images.

The rendered images are shown in Fig. 9. Fig. 9(a) is rendered with the original pyramid texture. Fig. 9(b) shows the image rendered with the interpolated representative value, which has only low frequency part of the original pyramid texture and the blending effects. Fig. 9(c) shows the image rendered by the proposed pyramid texture decompression.



(a)

(b)

(c)

Fig. 9. Rendered images. Rendered with (a) original pyramid texture, (b) the interpolated representative values and (c) the proposed pyramid texture decompression.

IV. Conclusion

We have proposed a texture compression scheme suitable for pyramid textures, which can be decompressed easily by simple hardware. The pyramid texture being used for mip-mapping is compressed using only one representative value databook, one residual vector codebook and one index map which are shared for all levels of the pyramid texture. The compression ratio of the proposed pyramid texture compression method is ranging from 10:1 to 40:1 according to the block size. The difference between the rendered image using the original pyramid texture and the image generated by pyramid texture decompression is so small that no visual difference can be found.

References

[1] L. Williams, "Pyramidal Parametrics," *Computer Graphics,* Vol. 17, No. 3, July 1983, pp. 1-11.

[2] Andrew C. Beers, Maneesh Agrawala, and Navin Chaddha, "Rendering from Compressed Textures", *Proceedings of SIGGRAPH 1996*, pp. 373-378.

[3] Anders Kugler, "High-Performance Texture Decompression Hardware", *The Visual Computer*, 1997, No.13, pp. 51-63.

[4] Ziyad S. Hakura and Anoop Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping,", *International Symposium on Computer Architecture*, 1997, pp. 108-120.

[5] Michael Cox, Narendra Bhandari, and Michael Shanta, "Multi-Level Texture Caching for 3D Graphics Hardware,", *International Symposium on Computer Architecture*, 1998, pp. 86-97.

[6] R. M. Gray, "Vector Quantization," *IEEE ASSP Magazine,* Apr. 1984.

[7] H. M. Hang and B. G. Haskell, "Interpolative Vector Quantization of Color Images," *IEEE Trans. Communications.*, COM-36, Apr. 1988, pp. 465-469.

[8] Paul S. Hecbert, "Survey of Texture Mapping," IEEE Computer Graphics and Applications, Vol. 6, pp. 56-67, 1986.