J. Wang and K. J. Dana: Compression of view dependent displacement maps.
In *Texture 2005: Proceedings of the 4th International Workshop on Texture Analysis and Synthesis*, pp. 143–148, 2005.

Compression of View Dependent Displacement Maps

Jing Wang and Kristin J. Dana
Electrical and Computer Engineering Department
Rutgers University
Piscataway, NJ 08854, USA
jingwang,kdana@caip.rutgers.edu

Abstract

Texture is important to enhance realism in surface representations for computer graphics and vision. View dependent displacement mapping (VDM) is proposed recently to give the effects of shadow, occlusion and silhouette, since traditional texture mapping can not produce all of these effects. Also, VDM can be viewed as a data structure to encode the texture geometry and exploit the modern graphics processing unit (GPU) to do interactive rendering. One drawback of the VDM approach is large memory consumption. Singular value decomposition (SVD) based compression has been proposed to alleviate this issue. In this paper, we present a method that combines k-means clustering and non-negative matrix factorization. Experimental evaluations show that the performance of our approach surpasses prior SVD-based method.

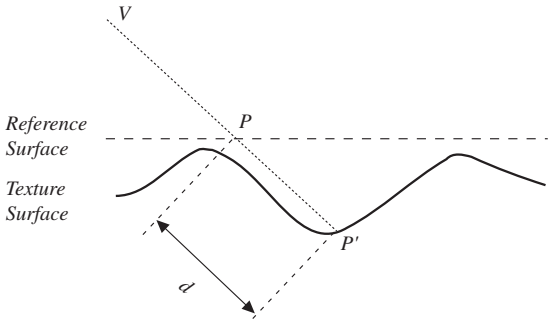

Figure 1. Illustration of VDM generation. From viewing direction V, we see the point $P^{'}$,instead of P. This displacement is called view dependent displacement, since it depends on the viewing direction V.

1 Introduction

Representing fine scale geometry can greatly enhance a surface representation. The ideal method should produce shadows, occlusions and silhouettes. Bump mapping, horizon mapping and displacement mapping are three classic methods to partially achieve this goal. Bump mapping [2] can produce bump visual effects, however it can not produce cast shadows, occlusions and silhouettes. Horizon mapping [13] is able to do self shadowing, but can not generate occlusions and edge silhouettes (the contour generated by the rough edge of the texture surface). Displacement mapping [4], while capable of occlusion and silhouette, does not produce shadow effects. There are also other texture mapping approaches that excel at different areas of texture rendering. By using warping techniques, relief texture mapping [16] can handle self-occlusion and silhouette, but self shadowing is missing. Also, simple parallax effect can be approximated by parallax mapping [7] but without effects of occlusions, silhouettes and self shadowing. As an alternative, bidirectional texture function (BTF) [5] is an image-based approach using texture images under different lighting and viewing conditions. The BTF has shadow and occlusion effects, but does not have geometry information.

Recently, view dependent displacement mapping (VDM) [20] is developed to have all three effects. Essentially,VDM can be regarded as a data structure to encode the fine scale geometry to do interactive rendering on the platform of modern graphics processing unit (GPU). The basic idea is illustrated in Figure 1. For each point P in the reference surface, with each viewing direction V, there is an intersection point $P^{'}$, that we actually see. The local geometry can be encoded as the distance d between P and $P^{'}$. It is obvious that d is dependent on the position of P and the viewing direction V. If P is in the edge of the texture, it may not have intersection point $P^{'}$, in this case, d can be assigned a value -1 as in [20]. This is important for generating the effect of the silhouette. A four dimensional texture function $d(x, y, \theta, \phi)$ is created, where x, y are the texture coordinates in the reference plane, and θ, ϕ are the polar and azimuth angle of the viewing direction V. In the original paper [20], curved surfaces are considered. For simplicity, we ignore curvature and assume a locally planar surface.

In the rendering phase, for each pixel, we have a viewing direction and lighting direction. And from the VDM, we can figure out the texture displacement, and do the silhouette determination and shadow determination. If $d(x, y, \theta, \phi)$ of a pixel is not -1, i.e. the line of sight in-

tersects the texture geometry, and this pixel is not in the shadow, we can assign a pixel value based on the shading model or image interpolation if sample views are available.

One obvious problem with using the VDM is the memory consumption. Suppose we have 128×128 samples in the spatial dimension, 32 polar angles and 16 in azimuth angles. Then the total amount of memory for storing the VDM is 8MB with each byte representing each $d(x, y, \theta, \phi)$. The compression is needed in order to make efficient use of the GPU texture memory. The SVD based approach in [20] can be summarized this way.

1. Maximal view polar angle map (MVM) is created as $\theta_{mvm}(x, y, \phi) = \max(\theta)$ such that $d(x, y, \theta, \phi) \neq -1$. Also, the -1 value of the corresponding polar viewing angle is replaced with $\theta_{mvm}(x, y, \phi)$.

2. VDM and MVM are reorganized into a 2D matrix, $A = [A_{vdm}\, A_{mvm}\,]$ where rows are indexed by x, y, ϕ, and columns are indexed by θ.

3. Applying SVD to A gives $A = U\lambda E^T$, and only small number of singular values in λ are needed to make approximation well.

There are some drawbacks of this approach. First, $d(x, y, \theta, \phi)$ is a positive number except for the special value -1, and it may be inappropriate to treat them altogether, although [20] also realizes this and creates the auxiliary MVM to alleviate this issue. Second, the arrangement of $A = [A_{vdm}\, A_{mvm}\,]$ can be used to exploit the fact that under different θ, $d(x, y, \theta, \phi)$ is in a low dimensional space, however it neglects the fact that texture in general is a local repetitive pattern. This property is better captured by our clustering based approach. Third, using SVD $A = U\lambda E^T$, U and E can have negative values, which will raise issues in hardware texture memory storage, and even worse, the approximated $d(x, y, \theta, \phi)$ can have negative value. Finally, the compression ratio is limited by rank of A, which in turn is limited by the number of sample values of θ as the columns of A are indexed by θ.

Let $D\,(x, y)$ be the VDM values for the samples of θ, ϕ associated with each x, y. For example, for 32 polar angles and 16 azimuth angles, $D\,(x, y)$ for a given x, y has 512 elements, and is treated as a 512 length vector. First we classify $D(x, y)$ into two categories. The first category includes $d(x, y, \theta, \phi)$ which has -1, that means it lies in the edge and may contribute to the silhouette effect. The second category includes $d(x, y, \theta, \phi)$ which always has positive values. For each of the two categories, we do k-means clustering, since we want to exploit the fact that texture is composed of repetitive patterns. For each cluster in the second category, we use non-negative matrix factorization approach to improve the approximation result. For the first category, since there is -1 value in $D(x, y)$, for each cluster we add 1 to make it non-negative, and then use non-negative matrix factorization. After we get the approximation, we simply make a threshold to get back these -1 value.

2 Related Work

For texture compression, there are many techniques developed in the image processing community. However, as [1] pointed out, texture compression in the context of rendering must meet some special requirements, such as decoding speed, random access, compression rate and visual quality, and encoding speed. The vector quantization method in [1] is suited for texture compression in terms of these special needs. In the context of VDM compression, encoding speed is not a critical issue, since VDM computing dominates the precomputing time.

Besides the traditional texture compression, bidirectional reflectance distribution function (BRDF) [15] and bidirectional texture function (BTF) [5] require an efficient representation to do real time rendering. For BRDF, homomorphic factorization in [14] was used for high performance rendering. While for BTF, [12] presented the concept of three dimensional textons to encode local appearance of the 3D texture. This concept has been exploited in [18] to synthesize bidirectional texture functions on arbitrary surfaces. Chained matrix factorization was adopted in [17] to make interactive rendering with hardware support. SVD based approach has also been used in BTF compression in [8]. However, none of these prior methods for BTF rendering incorporate geometry so they are limited in the quality of the texture prediction.

There are also many related work in the filed of image based rendering, for example, JPEG and MPEG are exploited in compression of lumigraph [6], more sophisticated method such as the multiple reference frame predication is provided in [21].

3 Method

Our method of compression is based on the observation that although there are infinite configurations in the real world texture, there are only finite configurations in the local scale, such as valleys, ramps, and edges, which is also exploited in [19]. VDM is essentially a local geometry encoding, so it is expected that $D\,(x, y)$ will form some clusters in high dimensional space. Because $D\,(x, y)$ can be -1 if there is no intersection between line of sight and texture geometry, it is quite natural to first classify $D\,(x, y)$ into two categories: $C^{(1)}$ and $C^{(2)}$. $D\,(x, y) \in C^{(1)}$ iff there exists $\theta^{'}$ and $\phi^{'}$, such that an element of $D\,(x, y)$ is -1. We denote these $D\,(x, y)$ as $D^{(1)}\,(x, y)$. $D\,(x, y) \in C^{(2)}$ iff there is no $\theta^{'}$ and $\phi^{'}$, such that $D\,(x, y) = -1$. We denote these $D\,(x, y)$ as $D^{(2)}\,(x, y)$. Then we can use the clustering methods to get the clusters of $C^{(1)}$ and $C^{(2)}$. Here, we use the k-means clustering for its simplicity and efficiency. After we do the clustering, we can assign the class labels to $D^{(1)}\,(x, y)$ and $D^{(2)}\,(x, y)$.

After clustering, there is still a lot of intraclass variation in each class. To efficiently reduce this variation, we choose the method of non-negative matrix factorization (NMF) [11], which also has been used in efficient

BRDF importance sampling in [9], and efficient representation and hardware rendering of surface light fields in [3]. For $D^{(1)}(x,y)$, since there is -1 value, we just use $D^{(1)}(x,y)+1$ in non-negative matrix factorization.

For each class l, it is straightforward to form the matrix $B^{(l)}$ with size $p\times q$, where each column of $B^{(l)}$ is a vector of $D(x,y)$ that comes from the same class l. We want to find non-negative matrix W with size $p\times m(m<<q)$ and H with size $m\times q$ to minimize Frobenius norm of $B^{(l)}-WH$, i.e.

$$W,H=\arg\min_{W,H}\|B^{(l)}-WH\|_F. \tag{1}$$

An iterative update method is provided in [11], namely

$$H_{i,j}=H_{i,j}\frac{(W^TB^{(l)})_{i,j}}{(W^TWH)_{i,j}}, \tag{2}$$

$$W_{i,j}=W_{i,j}\frac{(B^{(l)}H^T)_{i,j}}{(WHH^T)_{i,j}}. \tag{3}$$

There are couple of advantages using NMF instead of SVD. First, since W and H are non-negative, there is no worry about the negative values in storage and operation in GPU hardware. Second, as is shown in [10], NMF tends to learn the objects by their parts. It is always an additive operation. Since each column of $B^{(l)}$ is from the same class l, it is expected that they have similar local geometry. In other words, they may have close values of VDM under a subset of sampling viewing directions θ,ϕ. As a simple analogy, in [10], face images are decomposed into the eyes, mouths and other structures. In other words, since

$$B_k^{(l)}=\sum_{i=1}^{m}W_iH_{i,k},\ \ k\in 1,...,p \tag{4}$$

each column of $B^{(l)}$ can be considered as the addition of different structures represented by the columns W_i. Because W_i is non-negative, we can assume W_i (column of W) is also a valid measure of VDM under a subset of sampling viewing directions θ,ϕ. However in the SVD-based approach, since W_i can have negative values, it may not be a valid VDM at all. This feature will greatly reduce the noise in the rendering image based on NMF compared with SVD.

After we get the NMF, it can be directly used in approximate $D^{(2)}(x,y)$. However for $D^{(1)}(x,y)$, we need to put a threshold(0.5) to get back the -1 value. Our approach can be summarized in Figure 2.

Note that our proposed approach uses only linear arithmetic as in the SVD approach, with a small extra overhead in the index of the class labels. The compression ratio is analyzed as follow. (We ignore the special treatment for $D^{(1)}(x,y)$, i.e. adding 1 and threshold back to -1, since it does not affect the analysis of the compression ratio.)

Suppose each $D(x,y)$ has dimension $P=\Theta\times\Phi$ decided by azimuth and polar resolutions, and there are total $X\times Y$ (spatial resolutions) VDM vectors. We classify these

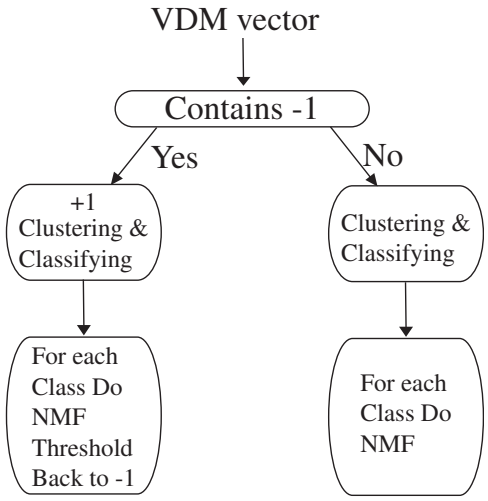

Figure 2. Flowchart of proposed VDM texture compression.

$X\times Y$ vectors to $C^{(1)}$ and $C^{(2)}$. $C^{(1)}$ has a size of $S^{(1)}$, and $C^{(2)}$ has a size of $S^{(2)}$, so $X\times Y=S^{(1)}+S^{(2)}$.

We have M clusters for $C^{(1)}$, and N clusters for $C^{(2)}$. For each $i\in 1,...,M$ cluster in $C^{(1)}$, denoted as $C_i^{(1)}$ with $S_i^{(1)}$ members. Clearly,

$$\sum_{i=1}^{M}S_i^{(1)}=S^{(1)}. \tag{5}$$

Suppose we create a matrix for each $C_i^{(1)}$, with $P\times S_i^{(1)}$, i.e. each column is a member of $C_i^{(1)}$, and approximate it with NMF, i.e. two non-negative matrix with size $P\times Q^{(1)}$, and $Q^{(1)}\times S_i^{(1)}$ respectively. Here we use a fixed $Q^{(1)}$. So the total size of the approximation is

$$\sum_{i=1}^{M}(P\times Q^{(1)}+Q^{(1)}\times S_i^{(1)})=P\times Q^{(1)}\times M+Q^{(1)}\times S^{(1)}. \tag{6}$$

Similarly, For each $j\in 1,...,N$ cluster in $C^{(2)}$, denoted as $C_j^{(2)}$ with $S_j^{(2)}$ members. Clearly,

$$\sum_{j=1}^{N}S_j^{(2)}=S^{(2)}. \tag{7}$$

Also, we create a matrix for each $C_j^{(2)}$, with $P\times S_j^{(2)}$, i.e. each column is a member of $C_j^{(2)}$, and approximate it with NMF, i.e. two non-negative matrix with size $P\times Q^{(2)}$, and $Q^{(2)}\times S_j^{(2)}$ respectively. Here we also use a fixed $Q^{(2)}$. So

the total size of the approximation is

$$\sum_{j=1}^{N}(P\times Q^{(2)}+Q^{(2)}\times S_j^{(2)}) = P\times Q^{(2)}\times N+Q^{(2)}\times S^{(2)}. \quad (8)$$

By Equation 6 and 8, the total compressed data is

$$P\times\ Q^{(1)}\times M+\ Q^{(1)}\times S^{(1)}+ \quad (9)$$

$$P\times\ Q^{(2)}\times N+\ Q^{(2)}\times S^{(2)} \quad (10)$$

$$=P\times(\ Q^{(1)}\times M+\ Q^{(2)}\times N)+ \quad (11)$$

$$Q^{(1)}\times S^{(1)}+Q^{(2)}\times S^{(2)}. \quad (12)$$

If $Q^{(1)}=Q^{(2)}=Q^{'}$, then the total compressed data is $P\times Q^{'}\times(M+N)+X\times Y\times Q^{'}$ independent of $S^{(1)}$ and $S^{(2)}$.

4 Experiment Results

We have done three experiments to verify our approach. In all of the examples, we use Θ =32 polar angles and Φ =16 azimuth angles, so the total is a $P=512=32\times16$ length vector for each VDM vector $D(x,y)$. The spatial resolution is $X=128, Y=128$. Thus, the total size of VDM data is $128\times128\times512$.

In the example of a random height texture in Figure 3, we have $S^{(1)}=14980$, $S^{(2)}=1404$, $Q^{(1)}=20$, $Q^{(2)}=16$, $M=32$, $N=16$. According to Equation 9, the compression ratio is about 10.74. Also, we use the SVD-based approach with 3 eigenvectors, thus the compressed ratio is about 10.67. They almost achieve the same compression ratio, but our visual result is much better.

Now, we go further in the compression. Suppose we only keep one eigenvector in SVD based approach, then the compression ratio is 32, which is a maximal compression ratio it can achieve. In our approach, we choose $Q^{(1)}=8$, $Q^{(2)}=8, M=16, N=16$, the same 32 compression ratio achieved. Figure 4 shows SVD based approach deteriorates quickly, however, our approach still has very good image quality. This fact shows great potential of our approach.

In all these examples, there is no attempt to optimize $Q^{(1)}, Q^{(2)}, M, N$ in order to get best compression ratio. Also, due to insufficient sampling and ignoring the curvature variation in the generation of VDM, there are some artifacts in the ground truth.

5 Summary and Future Work

In this paper, we present a clustering and NMF based approach for VDM texture compression. The experiment results show our method is better than SVD based approach. However, there is still room for improvement. For example, in our case, we use k-means clustering. It may not be appropriate to use this simple clustering in high dimensional space, and more sophisticated methods may be used. Also, an adaptive method to choose $Q^{(1)}, Q^{(2)}, M, N$ to get best compression result is another future research direction.

References

[1] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *SIGGRAPH '96*, pages 373–378, 1996.

[2] J. F. Blinn. Simulation of wrinkled surfaces. *ACM SIGGRAPH*, 12(3):286–292, 1978.

[3] W.-C. Chen, J.-Y. Bouguet, M. H. Chu, and R. Grzeszczuk. Light field mapping: efficient representation and hardware rendering of surface light fields. In *SIGGRAPH '02*, pages 447–456, 2002.

[4] R. L. Cook. Shade trees. In *SIGGRAPH '84*, pages 223–231, 1984.

[5] K. J. Dana, B. van Ginneken, S. K. Nayar, and J. J. Koenderink. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.

[6] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. *ACM SIGGRAPH*, pages 43–54, 1996.

[7] T. Kaneko and et al. Detailed shape representation with parallax mapping. *International Conference on Artificial Reality and Telexistence*, pages 205–208, 2001.

[8] M. L. Koudelka, S. Magda, P. N. Belhumeur, and D. J. Kriegman. Acquisition, compression and synthesis of bidirectional texture functions. *3rd International Workshop on Texture Analysis and Synthesis (Texture 2003)*, pages 59–64, 2003.

[9] J. Lawrence, S. Rusinkiewicz, and R. Ramamoorthi. Efficient brdf importance sampling using a factored representation. *ACM Trans. Graph.*, 23(3):496–505, 2004.

[10] D. D. Lee and H. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

[11] D. D. Lee and H. Seung. Algorithms for non-negative matrix factorization. *Adv. Neural Info. Proc. Syst.*, 13:556–562, 2001.

[12] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[13] N. Max. Horizon mapping: Shadows for bump-mapped surfaces. *The Visual Computer*, 4(2):109–117, 1988.

[14] M. D. McCool, J. Ang, and A. Ahmad. Homomorphic factorization of brdfs for high-performance rendering. In *SIGGRAPH '2001*, pages 171–178, 2001.

[15] F. E. Nicodemus, J. C. Richmon, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometic considerations and nomenclature for reflectance. *NBS Monograph 160*, October 1977.

[16] M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *SIGGRAPH '00*, pages 359–368, 2000.

[17] F. Suykens, K. Berge, A. Lagae, and P. Dutré. Interactive rendering with bidirectional texture functions. *Comput. Graph. Forum*, 22(3):463–472, 2003.

[18] X. Tong, J. Zhang, L.Liu, X.Wang, B. Guo, and H. Shum. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics*, 21(3):665–672, 2002.

[19] J. Wang and K. J. Dana. Hybrid textons: modeling surfaces with reflectance and geometry. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1:372–378, 2004.

[20] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. View-dependent displacement mapping. *ACM Transactions on Graphics*, 22(3):334–339, 2003.

[21] C. Zhang and J. Li. Compression of lumigraph with multiple reference frame (mrf) prediction and just-in-time rendering. *IEEE Data Compression Conference*, pages 254–263, March 2000.

Figure 3. The rendering results of different compression schemes. The first column uses the SVD based approach, and the second column is for proposed method, almost the same compression ratio(about 10.6) as the first column. The third column is for the uncompressed data. Our method introduces much less noise than the SVD based method, and keeps the silhouette well.

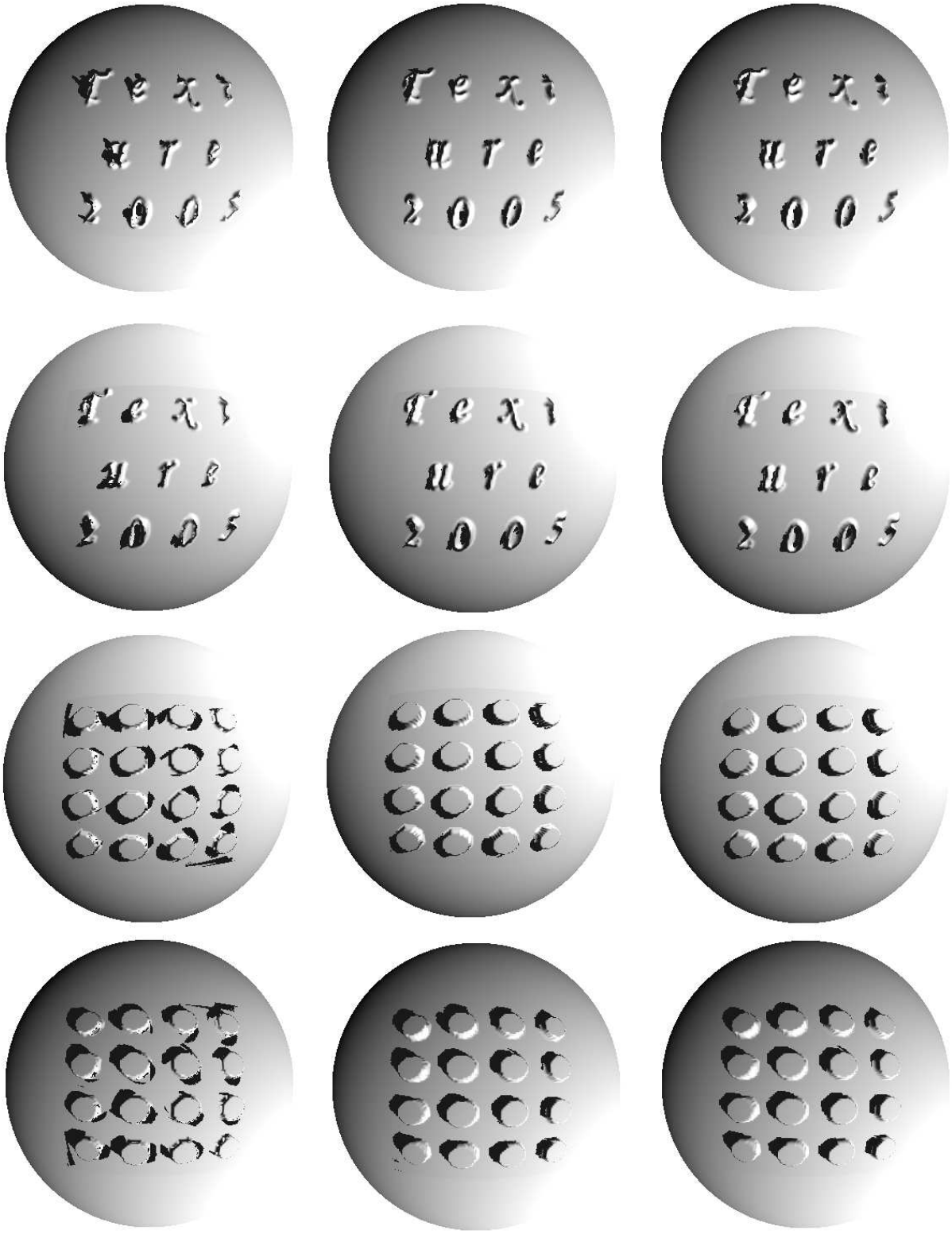

Figure 4. The rendering results of different compression schemes. The first column uses the SVD based approach, and the second column is for proposed method, the same compression ratio(32) as the first column. The third column is for the uncompressed data. With such high compression ratio, our method still works well, however SVD based approach becomes much worse.