Appears in ACM Transactions on Graphics (Special Issue for SIGGRAPH 2005)

Out-of-Core Tensor Approximation of Multi-Dimensional Matrices of Visual Data

Hongcheng Wang Qing Wu Lin Shi Yizhou Yu Narendra Ahuja

University of Illinois at Urbana-Champaign

Abstract

Tensor approximation is necessary to obtain compact multilinear models for multi-dimensional visual datasets. Traditionally, each multi-dimensional data item is represented as a vector. Such a scheme flattens the data and partially destroys the internal structures established throughout the multiple dimensions. In this paper, we retain the original dimensionality of the data items to more effectively exploit existing spatial redundancy and allow more efficient computation. Since the size of visual datasets can easily exceed the memory capacity of a single machine, we also present an out-of-core algorithm for higher-order tensor approximation. The basic idea is to partition a tensor into smaller blocks and perform tensor-related operations blockwise. We have successfully applied our techniques to three graphics-related data-driven models, including 6D bidirectional texture functions, 7D dynamic BTFs and 4D volume simulation sequences. Experimental results indicate that our techniques can not only process out-of-core data, but also achieve higher compression ratios and quality than previous methods.

CR Categories: I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.7 [**Computer Graphics**]: Three-dimensional Graphics and Realism—color, shading, shadowing, and texture

Keywords: Multilinear Models, Spatial Coherence, Block-Based Partitioning, Bidirectional Texture Functions, Volume Simulations

1 Introduction

With the advent of data-driven models, such as light fields [Levoy and Hanrahan 1996], bidirectional texture functions (BTFs) [Dana et al. 1999], and reflectance fields, graphics researchers have been striving for powerful representation and compression methods to cope with the enormous amounts of data involved. Multilinear models based on tensor approximation have caught researchers' attention recently. Such models can not only reduce the amount of data and dimensionality, but also learn a compact multilinear representation. In practice, they have proven to be more effective than traditional dimensionality reduction methods, such as principal component analysis (PCA).

In multilinear modeling of multi-dimensional, multi-modal data, a crucial problem is determining the order of the tensor. For example, given a collection of images as the input dataset, one typical strategy is to convert each image into a large column vector with a length equal to the number of pixels. This length is fixed during subsequent tensor approximation. Such a strategy is actually suboptimal because it ignores the large amount of redundancy due to spatial coherence within every multi-dimensional data item. This is obvious for images as adjacent rows or columns of an image often exhibit similar color patterns. In this paper, we propose to formulate higher-order tensors to exploit spatial coherence available in individual multi-dimensional data items, such as 2D images and 3D volumes. As a result, subsequent tensor approximation can also remove the spatial redundancy in these data and give rise to a much more compact representation.

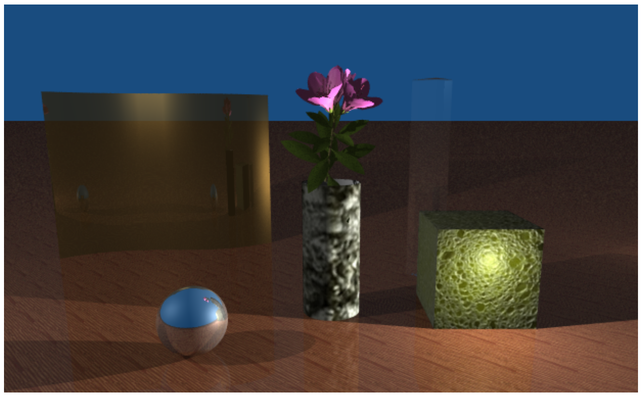


Figure 1: A virtual scene with a cube mapped with a SPONGE BTF and a vase mapped with a LICHEN BTF. There is a point light source near the "sponge".

Since the most important motivation of tensor approximation is to produce a compact representation from a huge amount of redundant data, it becomes particularly meaningful when the original data cannot fit into the main memory of a single machine. Virtual memory cannot help much because the largest possible (virtual) memory capacity of a computer with a 32-bit architecture is only around 4GB and multi-dimensional datasets larger than that are not uncommon. Out-of-core processing capability is therefore much desired. Such a technique is useful even when the virtual memory allows addressing of the entire data but, in practice, the small physical memory will cause extensive swapping. Typically, application-specific data partitioning and swapping can give far better performance. We present an efficient out-of-core algorithm for tensor approximation. It is based on a simple block-based partition. With such an algorithm, we are able to process datasets larger than 10GB on a PC with less than 1GB memory.

We have successfully applied our new tensor approximation techniques to three multi-dimensional data-driven models, including 6D bidirectional texture functions, 7D temporally varying BTFs (dynamic BTFs) and 4D volume simulation sequences. Learned multilinear models for these multi-dimensional datasets become sufficiently small and can fit into the main memory of a single machine for rendering or visualization.

2 Background and Related Work

2.1 Related Work

A traditional dimensionality reduction technique is principle component analysis (PCA), which has been widely used and extended in computer graphics, including for surface light field representation [Nishino et al. 1999; Chen et al. 2002], BTF representation [Koudelka et al. 2003; Sattler et al. 2003; Liu et al. 2004], reflectance modeling [Matusik et al. 2003], deformation database compression [James and Fatahalian 2003], and many other applications. PCA is used to find a set of mutually orthogonal basis functions which capture the largest variation in the training data. It is typically computed through singular value decomposition (SVD). For large datasets, incremental or out-of-core algorithms [Rabani and Toledo 2001; Brand 2002] have been developed for SVD.

Multilinear algebra has recently received much attention in computer vision and signal processing. N-mode SVD was discussed in [Lathauwer et al. 2000a], and has been used in computer vision applications such as face recognition [Vasilescu and Terzopoulos 2002] and facial expression decomposition [Wang and Ahuja 2003]. Multilinear modeling has been further applied to bidirectional texture functions in [Furukawa et al. 2002; Vasilescu and Terzopoulos 2004] where compact BTF models were built for image-based rendering using a higher-order tensor approximation algorithm, such as the one in [Lathauwer et al. 2000b]. Nevertheless, it was mentioned in [Vasilescu and Terzopoulos 2004] that, for the same compression ratio, the Root Mean Squared Error (RMSE) of their tensor approximation is higher than PCA.

All these PCA- or tensor-based techniques adopt the image-as-vector representation by concatenating image rows into a single vector. One inherent problem of this representation is that the spatial redundancy within each image matrix is not fully utilized, and some information on local spatial relationships is lost. Realizing the problem of image-as-vector formulation, some researchers in computer vision and machine learning have recently begun to treat an image as a matrix [Shashua and Levin 2001; Yang et al. 2004; Ye 2004]. Although these methods are reasonably effective on collections of 2D images, it is hard to generalize these methods to higher-dimensional datasets such as a collection of 3D volumes. In addition, the decompositions adopted in these methods are not as powerful as the decomposition in [Lathauwer et al. 2000b]. The method we present in this paper is based on [Lathauwer et al. 2000b], and can be applied to multi-modal datasets of arbitrarily high dimensions.

2.2 Rank-$(R_1, R_2, ..., R_N)$ Approximation of High-Order Tensors

Overview of Multilinear Algebra. In the notation we use, matrices are denoted by bold capitals $(\mathbf{A}, \mathbf{B}, \cdots)$, and tensors by calligraphic letters $(\mathcal{A}, \mathcal{B}, \cdots)$.

A tensor is a higher order generalization of a vector (1st-order tensor) and a matrix (2nd-order tensor). An Nth-order tensor is denoted as: $\mathcal{A} \in \Re^{I_1 \times I_2 \times \ldots \times I_N}$. An element of $\mathcal{A}$ is denoted as $a_{i_1 \ldots i_n \ldots i_N}$, where $1 \leq i_n \leq I_n$. A mode-n vector is a column vector consisting of the following set of elements of $\mathcal{A}$: $\{a_{i_1 \ldots i_n \ldots i_N}\}_{i_n=1}^{I_n}$, where the index i_n varies while the others have fixed values. Unfolding a tensor $\mathcal{A}$ along the nth mode is denoted as *uf(𝒜, n)*, which results in a matrix, $\mathbf{A}_{(n)} \in \Re^{I_n \times (I_1 I_2 \ldots I_{n-1} I_{n+1} \ldots I_N)}$, consisting of all the mode-n vectors. The mode-n product of a tensor $\mathcal{A}$ and a matrix $\mathbf{U} \in^{J_n \times I_n}$, denoted by $\mathcal{A} \times_n \mathbf{U}$, is defined as a tensor with entries: $(\mathcal{A} \times_n \mathbf{U})_{i_1 \ldots i_{n-1} j_n i_{n+1} \ldots i_N} = \sum_{i_n} a_{i_1 \ldots i_N} u_{j_n i_n}$. The scalar product of two tensors $\mathcal{A}, \mathcal{B} \in \Re^{I_1 \times I_2 \times \cdots \times I_N}$ is defined as: $\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_N} a_{i_1 i_2 \cdots i_N} b_{i_1 i_2 \cdots i_N}$. The Frobenius norm of a tensor $\mathcal{A} \in \Re^{I_1 \times I_2 \times \cdots \times I_N}$ is then defined as $\|\mathcal{A}\| = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}$. The n-rank of $\mathcal{A}$, denoted by $R_n = rank_n(\mathcal{A})$, is the dimension of the vector space spanned by the mode-n vectors. Refer to [Lathauwer et al. 2000a] for more details on multilinear algebra.

N-mode SVD [Lathauwer et al. 2000a] performs regular SVD on every matrix $\mathbf{A}_{(n)}$ resulted from unfolding the nth mode of tensor $\mathcal{A}$. Each such regular SVD returns a mode-n basis matrix, $\mathbf{U}^{(n)}$. As a result, $\mathcal{A}$ can be expressed as the product: $\mathcal{A} = \mathcal{Z} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times \cdots \times_N \mathbf{U}^{(N)}$, where, $\mathcal{Z}$ is an $I_1 \times I_2 \times \ldots \times I_N$ core tensor, and $\mathbf{U}^{(n)} = (\mathbf{U}_1^{(n)} \mathbf{U}_2^{(n)} \ldots \mathbf{U}_{I_n}^{(n)})$ is a unitary $(I_n \times I_n)$ basis matrix, the columns of which span the I_n-dimensional vector space where the mode-n vectors belong.

Rank-$(R_1, R_2, ..., R_N)$ Approximation. Given a real Nth-order tensor $\mathcal{A} \in \Re^{I_1 \times I_2 \times \ldots \times I_N}$, rank-$(R_1, R_2, ..., R_N)$ approximation of $\mathcal{A}$ is formulated as finding a lower-rank tensor $\tilde{\mathcal{A}} \in \Re^{I_1 \times I_2 \times \ldots \times I_N}$ with $\text{Rank}_1(\tilde{\mathcal{A}}) = R_1 \leq \text{Rank}_1(\mathcal{A})$, $\text{Rank}_2(\tilde{\mathcal{A}}) = R_2 \leq \text{Rank}_2(\mathcal{A})$, ..., $\text{Rank}_N(\tilde{\mathcal{A}}) = R_N \leq \text{Rank}_N(\mathcal{A})$, such that the following least-squares cost function is minimized:

$$\tilde{\mathcal{A}} = \arg\min_{\hat{\mathcal{A}}} \left\| \mathcal{A} - \hat{\mathcal{A}} \right\|^2. \tag{1}$$

The desired tensor is represented as:

$$\tilde{\mathcal{A}} = \mathcal{B} \times_1 \mathbf{U}^{(1)} \times_2 \mathbf{U}^{(2)} \times \cdots \times_N \mathbf{U}^{(N)}, \tag{2}$$

where $\mathbf{U}^{(1)} \in \Re^{I_1 \times R_1}$, $\mathbf{U}^{(2)} \in \Re^{I_2 \times R_2}$, $\cdots$, $\mathbf{U}^{(N)} \in \Re^{I_N \times R_N}$ and $\mathcal{B} \in \Re^{R_1 \times R_2 \times \cdots \times R_N}$. $\mathbf{U}^{(i)}$ has orthonormal columns for $1 \leq i \leq N$. When $R_1, R_2, \ldots, R_N$ are sufficiently small, the core tensor $\mathcal{B}$ and the basis matrices, $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \cdots, \mathbf{U}^{(N)}$, together give rise to a compact approximation of the original tensor $\mathcal{A}$. In this paper, we seek such compact approximations for multi-dimensional visual data.

Given basis matrices, $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \cdots, \mathbf{U}^{(N)}$, $\mathcal{B}$ can be obtained as $\mathcal{B} = \mathcal{A} \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \cdots \times_N \mathbf{U}^{(N)^T}$. Therefore, only the basis matrices are the unknowns in this optimization problem. The Alternative Least Square (ALS) is used in [Tucker 1966; Kroonenberg and de Leeuw 1980; Lathauwer et al. 2000b] to find a (locally) optimal solution of (1). In each iterative step, it optimizes only one of the basis matrices, while keeping others fixed. For example, with $\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(n-1)}, \mathbf{U}^{(n+1)}, \cdots, \mathbf{U}^{(N)}$ fixed, we project tensor $\mathcal{A}$ onto the $(R_1, \cdots, R_{n-1}, R_{n+1}, \cdots, R_N)$ - dimensional space, i.e., $\mathbf{U}_{j+1}^{(n)} = \mathcal{A} \times_1 \mathbf{U}_{j+1}^{(1)^T} \times \cdots \times_{n-1} \mathbf{U}_{j+1}^{(n-1)^T} \times_{n+1} \mathbf{U}_j^{(n+1)^T} \times \cdots \times_N \mathbf{U}_j^{(N)^T}$. Then the columns of $\mathbf{U}^{(n)}$ can be found as the first R_n columns of the left singular matrix of $uf(\mathbf{U}_{j+1}^{(n)}, n)$ using singular value decomposition. These steps are summarized in Algorithm 1.

One implementation issue is the initialization of the algorithm. In the algorithm presented in [Lathauwer et al. 2000b] as well as in [Vasilescu and Terzopoulos 2004], the values of $\mathbf{U}^{(n)}$ $(1 \leq n \leq N)$ were initialized with the truncated basis matrices obtained from N-mode SVD. However, the computation of N-mode SVD is expensive in terms of both space and time complexity, and is cumbersome for out-of-core calculation. Instead, we use $\mathbf{U}_0^{(n)} = \begin{bmatrix} \mathbf{I}_{R_n} & 0 \end{bmatrix}^T$, where $\mathbf{I}_{R_n}$ is the $R_n \times R_n$ identity matrix, or $\mathbf{U}_0^{(n)}$ = uniformly distributed random numbers (though columns are not necessarily orthonormal). Empirically, we have not seen any major difference in the results obtained using these initializations vs. those obtained from N-mode SVD. Our initializations are much simpler and faster to compute.

Algorithm 1: Rank-($R_1, R_2, \cdots, R_N$) Tensor Approximation

```
Data: Given $\mathcal{A}$ and $R_1, R_2, \cdots, R_N$
Result: Find $\mathbf{U}^k, (1 \le k \le N)$ and $\mathcal{B}$.

Initialize $\mathbf{U}_0^{(k)} \in \Re^{I_k \times R_k}, 1 \le k \le N$;
while $\sim stop$ do
    $\tilde{\mathbf{U}}_{j+1}^{(1)} = uf(\mathcal{A} \times_2 \mathbf{U}_j^{(2)^T} \times_3 \mathbf{U}_j^{(3)^T} \cdots \times_N \mathbf{U}_j^{(N)^T}, 1)$;
    $\mathbf{U}_{j+1}^{(1)} = SVD(\tilde{\mathbf{U}}_{j+1}^{(1)}, R_1)$;
    $\tilde{\mathbf{U}}_{j+1}^{(2)} = uf(\mathcal{A} \times_1 \mathbf{U}_{j+1}^{(1)^T} \times_3 \mathbf{U}_j^{(3)^T} \cdots \times_N \mathbf{U}_j^{(N)^T}, 2)$;
    $\mathbf{U}_{j+1}^{(2)} = SVD(\tilde{\mathbf{U}}_{j+1}^{(2)}, R_2)$;
    $\cdots$
    $\tilde{\mathbf{U}}_{j+1}^{(N)} =$
    $uf(\mathcal{A} \times_1 \mathbf{U}_{j+1}^{(1)^T} \times_2 \mathbf{U}_{j+1}^{(2)^T} \cdots \times_( N-1)\mathbf{U}_{j+1}^{(N-1)^T}, N)$;
    $\mathbf{U}_{j+1}^{(N)} = SVD(\tilde{\mathbf{U}}_{j+1}^{(N)}, R_N)$;
    $\mathcal{B} = \mathcal{A} \times_1 \mathbf{U}_{j+1}^{(1)^T} \times_2 \mathbf{U}_{j+1}^{(2)^T} \times \cdots \times_N \mathbf{U}_{j+1}^{(N)^T}$;
    $if(\|\mathcal{B}_{j+1}\|^2 - \|\mathcal{B}_j\|^2 < \varepsilon)\ stop$
end
```

3 Out-of-Core Higher-Order Tensor Approximation

In this section we develop an out-of-core tensor approximation algorithm. The input to our algorithm is a multi-dimensional multi-modal dataset. Each data item in the dataset is assumed to be defined on a multi-dimensional rectangular region, such as a 2D rectangular image or a 3D volume grid. The output from our algorithm is a rank-($R_1, R_2, \cdots, R_N$) tensor approximation, including a core tensor $\mathcal{B} \in \Re^{R_1 \times R_2 \times \cdots \times R_N}$ and the truncated basis matrices, $\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \cdots, \mathbf{U}^{(N)}$, if the input dataset is converted to an Nth-order tensor, $\mathcal{A} \in \Re^{I_1 \times I_2 \times \ldots \times I_N}$.

There are three intermediate steps our algorithm needs to perform before generating an output: first, an effective tensor representation should be chosen for the input dataset so that spatial as well as other redundancies can be removed in later stages; second, the resulting tensor needs to be partitioned into sufficiently small blocks stored on the hard disks so that each block can fit into the main memory of a single machine; third, Algorithm 1 needs to be adapted so that it can take the blocks of the original tensor, and still output the correct approximation as if the tensor for the original dataset had not been partitioned.

3.1 Tensor Construction

As mentioned previously, each input data item is already multi-dimensional. Thus, a basic element, such as a pixel, in the data item has two direct neighbors along each dimension. Since real data exhibit spatial coherence unlike pure noise, the difference between an element and its neighbors are typically not substantial. Adjacent rows or columns of elements are often similar. To exploit this type of spatial redundancy in tensor approximation, we preserve the original neighborhood structure when converting the input dataset into a tensor. As a result, each data item is considered as a subtensor of the tensor representing the entire dataset. For example, every 2D image in an image database is already treated as a second-order subtensor, and every 3D volume in a volume database is considered as a third-order subtensor. For simple image ensembles without specific modes, we use third-order tensors, $\mathcal{A}_{ensemble} \in \Re^{I_1 \times I_2 \times I_3}$, where $I_1 \times I_2$ is the dimensionality of each image, and I_3 is the number of images. However, if an image ensemble has two other modes, such as lighting directions and viewing directions for a BTF, it should be represented as a fourth-order tensor.

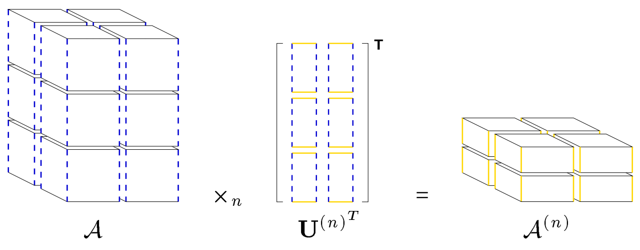


Figure 2: Block-based mode-n product between a 3rd-order tensor $\mathcal{A}$ and its transposed basis matrix $\mathbf{U}^{(n)^T}$. $\mathcal{A}$ is vertically partitioned into three segments (dashed blue) while $\mathbf{U}^{(n)}$ is vertically partitioned into three (dashed blue) and horizontally partitioned into two segments (solid gold). In the resulting tensor $\mathcal{A}^{(n)}$, there are only two vertical segments (solid gold). Each block in $\mathcal{A}^{(n)}$ is obtained using (3).

Although our tensor construction scheme retains the original dimensionality of images or volumes, it actually implicitly encodes the covariance between any pair of elements (pixels or voxels) in the image or volume space. Let us consider the aforementioned image ensemble which has I_3 images with a resolution $I_1 \times I_2$. If we represent each image as a vector as in PCA, the entire image ensemble is represented as a matrix $\mathbf{B} \in \Re^{(I_1 I_2) \times I_3}$. On the other hand, if we represent each image as a matrix, we need to factorize three unfolded matrices, i.e. $\mathbf{A}_{(1)} \in \Re^{I_1 \times (I_2 I_3)}$, $\mathbf{A}_{(2)} \in \Re^{I_2 \times (I_3 I_1)}$, and $\mathbf{A}_{(3)} \in \Re^{I_3 \times (I_1 I_2)}$. Note that $\mathbf{A}_{(3)}$ is actually the transpose of $\mathbf{B}$. Suppose $\mathbf{U}_i$ is an eigenvector of the covariance matrix of $\mathbf{B}$. Then $\mathbf{A}_{(3)}\mathbf{U}_i$ is actually an eigenvector of the covariance matrix of $\mathbf{A}_{(3)}$. Because of this relationship between $\mathbf{A}_{(3)}$ and $\mathbf{B}$, our approach indirectly encodes the covariance between an arbitrary pair of pixels in the image plane. More importantly, our scheme also encodes row and column covariances by factorizing $\mathbf{A}_{(1)}$ and $\mathbf{A}_{(2)}$.

Another major advantage of this tensor construction is that the size of the basis matrices for tensor approximation becomes much smaller because an image or volume is not converted to a vector any more. Smaller basis matrices make corresponding SVD in Algorithm 1 computationally much more tractable.

3.2 Block-Based Tensor Partitioning

In this section, we consider a tensor $\mathcal{A}$ as an N-dimensional array with size I_i along the n-th index. Suppose $\mathcal{A}$ needs to be partitioned into M blocks. We require that each block remains as an Nth-order subtensor. Thus, we partition the ranges of $\mathcal{A}$'s indices. If the range of the n-th ($1 \le n \le N$) index is evenly partitioned into P_n segments, each segment has an approximate size of $\lfloor I_n/P_n \rfloor$. For notational simplicity, we assume I_n/P_n is an integer. Then each block is a tensor itself in $\Re^{I_1/P_1 \times I_2/P_2 \times \ldots \times I_N/P_N}$. As long as $M = \prod_n P_n$, the specific values of $\{P_n\}_{n=1}^N$ are flexible and should be determined according to the values of $\{I_n\}_{n=1}^N$. If a block consists of elements occupying the p_n-th ($1 \le n \le N, 1 \le p_n \le P_n$) segment along the n-th index of $\mathcal{A}$, we denote that block as $\mathcal{A}_{[p_1,p_2,\ldots,p_N]}$.

Converting the input dataset of our algorithm from its original format into this block-structured tensor format is straightforward, and needs to be performed as a preprocessing step. The resulting blocks should be saved into separate files on hard disks to allow efficient random access to all of $\mathcal{A}$'s elements, which is required by Algorithm 1.

Once tensor $\mathcal{A}$ has been partitioned, the basis matrices,

$\mathbf{U}^{(1)}, \mathbf{U}^{(2)}, \cdots, \mathbf{U}^{(N)}$, should be partitioned accordingly as follows. Since $\mathbf{U}^{(n)} \in \Re^{I_n \times R_n}$, the range of its first index, I_n, should still be partitioned into P_n segments. If the range of its second index, R_n, is partitioned into Q_n segments, then Q_n should satisfy $R_n/Q_n \leq I_n/P_n$, to ensure that the blocks do not increase their size after mode-n products so that they can still fit into the memory. We use $\mathbf{U}^{(n)}_{[p,q]}$ to denote the block of $\mathbf{U}^{(n)}$ that occupies the p-th segment along its first index and q-th segment along its second index. Such block-based partitioning of tensors and basis matrices is visualized in Fig. 2.

3.3 Block-Based Tensor Approximation

Our block-based out-of-core algorithm was designed by adapting Algorithm 1 to operate on the block structures of the input tensor and the basis matrices. The blocks of the input tensor are stored on hard disks. Since the basis matrices are fairly small, all the blocks of the basis matrices stay in memory all the time. There are two essential operations in Algorithm 1 that can pose potential difficulties for an out-of-core algorithm: the first one is computing a sequence of consecutive mode-n tensor-matrix products like $\mathcal{A} \times_2 \mathbf{U}_j^{(2)^T} \times_3 \mathbf{U}_j^{(3)^T} \cdots \times_N \mathbf{U}_j^{(N)^T}$; the second is running SVD on an out-of-core matrix with a large number of columns, such as $SVD(\tilde{\mathbf{U}}^{(1)}_{j+1}, R_1)$ where $\tilde{\mathbf{U}}^{(1)}_{j+1}$ has not been truncated, and the number of its columns is $R_2 \times R_3 \times \cdots \times R_N$. In the following, we will focus on designing out-of-core steps for these two types of operations.

Our out-of-core implementation of a mode-n tensor-matrix product was inspired by block-based matrix-matrix multiplications. As shown in Fig. 2, the outcome of the mode-n product between a tensor $\mathcal{A}$, with $P_1 \times P_2 \times \cdots \times P_n \times \cdots \times P_N$ blocks, and a transposed basis matrix $\mathbf{U}^{(n)^T}$, with $Q_n \times P_n$ blocks, is still a tensor $\mathcal{A}^{(n)}$, but with $P_1 \times P_2 \times \cdots \times Q_n \times \cdots \times P_N$ blocks. We compute the blocks of $\mathcal{A}^{(n)}$ sequentially. The block of $\mathcal{A}^{(n)}$ denoted by $\mathcal{A}^{(n)}_{[p_1,p_2,\ldots,q_n,\ldots,p_N]}$ is computed as follows.

$$\mathcal{A}^{(n)}_{[p_1,p_2,\ldots,q_n,\ldots,p_N]} = \sum_{p_n=1}^{P_n} \mathcal{A}_{[p_1,p_2,\ldots,p_n,\ldots,p_N]} \times_n \mathbf{U}^{(n)^T}_{[p_n,q_n]}, \tag{3}$$

where each blockwise mode-n product, such as $\mathcal{A}_{[p_1,p_2,\ldots,p_n,\ldots,p_N]} \times_n \mathbf{U}^{(n)^T}_{[p_n,q_n]}$, is computed by loading the block $\mathcal{A}_{[p_1,p_2,\ldots,p_n,\ldots,p_N]}$ into the memory, and performing the product in-core; the summation loop $\sum_{p_n=1}^{P_n}$ is implemented by adding the result of $\mathcal{A}_{[p_1,p_2,\ldots,p_n,\ldots,p_N]} \times_n \mathbf{U}^{(n)^T}_{[p_n,q_n]}$ to another block representing the partial sum. The partial sum resides in the memory until the summation loop has been completed. It is then saved to the hard disk. The number of disk I/O's is $P_n + 1$ for computing $\mathcal{A}^{(n)}_{[p_1,p_2,\ldots,q_n,\ldots,p_N]}$. Since $\mathcal{A}^{(n)}$ has MQ_n/P_n blocks, the total number of disk I/O's is $MQ_n(P_n+1)/P_n$. Once we know how to implement one mode-n product, implementing a sequence of consecutive mode-n products becomes straightforward by adding an outer loop.

Performing SVD on an out-of-core matrix $\mathbf{A} \in \Re^{s \times t}$ can either follow an incremental algorithm, such as the one in [Brand 2002], or an out-of-core algorithm, such as the one in [Rabani and Toledo 2001]. Nevertheless, there exists a simple strategy that can avoid the invocations of such algorithms most of the time. In our context, $\mathbf{A}$ is an unfolded tensor, which means that $s \ll t$ because s represents the size of one dimension while t is the product of the sizes of multiple dimensions. Suppose the SVD of $\mathbf{A}$ is $\mathbf{USV}^T$. To avoid the large value of t, we exploit the fact that the SVD of

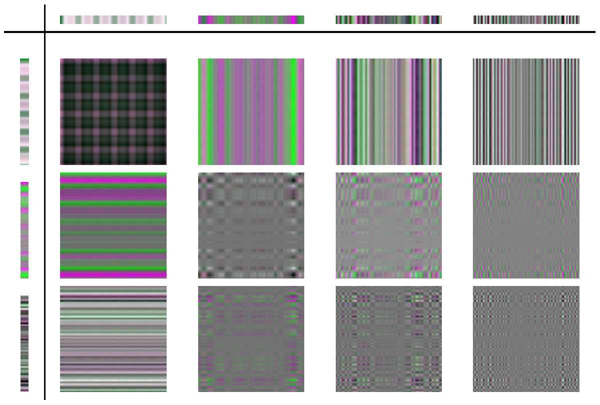


Figure 3: Basis images created from the basis matrices, $\mathbf{U}^{row}$ and $\mathbf{U}^{col}$ in (4). Each image is the tensor product of a pair of column vectors from these two matrices.

$\mathbf{AA}^T$ is $\mathbf{US^2U}^T$. Note that in Algorithm 1, we only need a subset of the column vectors from the left singular matrix, $\mathbf{U}$, of $\mathbf{A}$. Therefore, once we have obtained $\mathbf{U}$ from the SVD of $\mathbf{AA}^T$, it is not necessary to compute $\mathbf{V}$ anymore. Furthermore, due to our tensor construction scheme (Section 3.1), the size of $\mathbf{AA}^T \in \Re^{s \times s}$ is small. In our experiments, the value of s is usually only up to a few hundred. Thus, the SVD of $\mathbf{AA}^T$ can be computed in-core. However, before SVD, the matrix product $\mathbf{AA}^T$ needs to be computed blockwise since $\mathbf{A}$ is large and out-of-core.

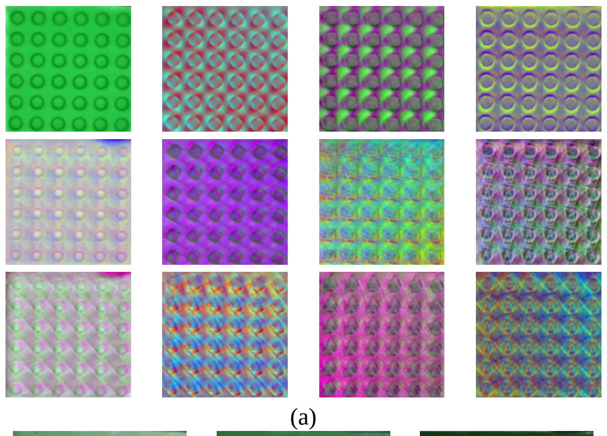
4 Applications

We apply the out-of-core tensor approximation algorithm developed in the previous section to three different data-driven models to demonstrate the usefulness of this algorithm.

4.1 BTF modeling and compression

The bidirectional texture function or BTF captures the appearance of extended textured surfaces. A BTF is defined as a six dimensional function with a 2D texture associated with every possible combination of illumination and view directions which account for the other four dimensions. When the illumination or view direction changes continuously, the observed 2D texture also changes smoothly, which suggests redundancy in the illumination and view spaces in addition to the spatial redundancy in the 2D texture image. Recent work on BTFs focuses on acquisition, representation, rendering and synthesis [Dana et al. 1999; Leung and Malik 1999; Liu et al. 2001; Furukawa et al. 2002; Tong et al. 2002; Han and Perlin 2003; Koudelka et al. 2003; Sattler et al. 2003; Vasilescu and Terzopoulos 2004; Yu and Chang 2005].

Since the BTF is originally represented as a large collection of images, we compute a high-order tensor with reduced ranks as a compact generative model, which captures the essential characteristics while removing the redundancies. As mentioned in Section 3.1, we organize a BTF as a fourth-order tensor, $\mathcal{A}_{BTF} \in \Re^{I_{row} \times I_{col} \times I_{illum} \times I_{view}}$, where I_{row} and I_{col} are the numbers of rows and columns of each image, and I_{illum} and I_{view} are the numbers of illumination and view directions, respectively. Although the view or illumination space is two-dimensional, we do not retain the two dimensions because unlike the number of pixels in an image row or column, the number of view or lighting directions along each dimension is quite small, typically below 40. It is hard to achieve a high compression ratio on such a small set. When the samplings



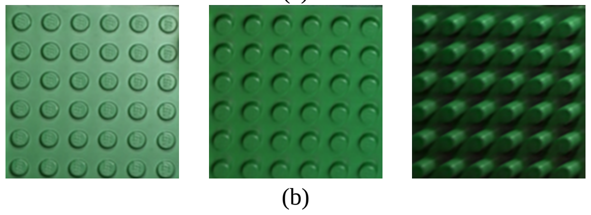


Figure 4: (a) A second set of basis images obtained by transforming the basis images in Fig. 3 by the core tensor $\mathcal{B}_{BTF}$. These are basis images for the view and illumination modes. (b) Examples of reconstructed images.

of such directions become much denser, retaining the original two dimensions would likely achieve better compression ratios.

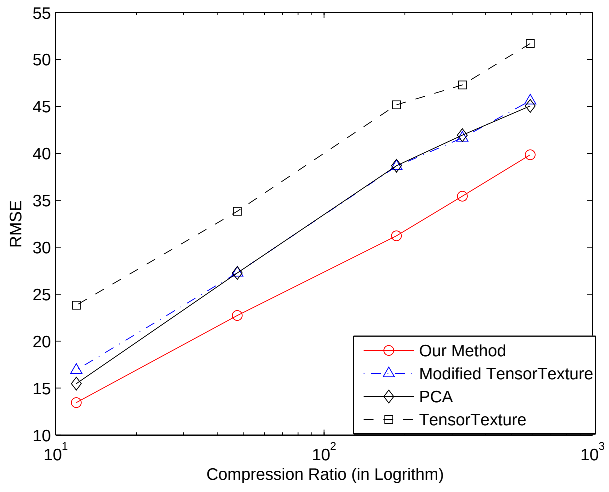
To remove redundancy and produce a compact representation, we use another fourth-order tenser $\tilde{\mathcal{A}}_{BTF}$ with reduced ranks to approximate $\mathcal{A}_{BTF}$ as in Section 2.2. The new tensor is represented as:

$$\tilde{\mathcal{A}}_{BTF} = \mathcal{B}_{BTF} \times_1 \mathbf{U}^{row} \times_2 \mathbf{U}^{col} \times_3 \mathbf{U}^{illum} \times_4 \mathbf{U}^{view}, \quad (4)$$

where $\mathbf{U}^{row} \in \Re^{I_{row} \times R_{row}}$, $\mathbf{U}^{col} \in \Re^{I_{col} \times R_{col}}$, $\mathbf{U}^{illum} \in \Re^{I_{illum} \times R_{illum}}$, $\mathbf{U}^{view} \in \Re^{I_{view} \times R_{view}}$, and $\mathcal{B}_{BTF} \in \Re^{R_{row} \times R_{col} \times R_{illum} \times R_{view}}$. $R_{row}, R_{col}, R_{illum}$ and R_{view} specify the reduced ranks (columns) of the four basis matrices for the four modes. The reduced ranks are provided by the user to control the approximation error in (1). The smaller they are, more compact the resulting new tensor is. Given the reduced ranks, the out-of-core tensor approximation algorithm in Section 3.3 is applied to solve for the reduced basis matrices and the core tensor, $\mathcal{B}_{BTF}$. The reduced basis matrices and the core tensor together comprise the compact representation.

The process of reconstructing a BTF image from our 4th-order tensor is outlined in Figs. 3 and 4. As shown in Fig. 3, if we take one column vector $\mathbf{u}_r$ from $\mathbf{U}^{row}$ and another column vector $\mathbf{u}_c$ from $\mathbf{U}^{col}$, the tensor between them, $\mathbf{u}_r\mathbf{u}_c^T$, defines a basis image. All such basis images together interact with the core tensor, $\mathcal{B}_{BTF}$, to produce a different set of basis images, shown in Fig. 4(a), for different view and illumination directions. Mathematically, the second set of basis images are obtained by performing $\mathcal{C}_{BTF} = \mathcal{B}_{BTF} \times_1 \mathbf{U}^{row} \times_2 \mathbf{U}^{col}$ where $\mathcal{C}_{BTF} \in \Re^{I_{row} \times I_{col} \times R_{illum} \times R_{view}}$. Then, if we take a pair of row vectors, $\mathbf{l}$ and $\mathbf{v}$, from $\mathbf{U}^{illum}$ and $\mathbf{U}^{view}$, respectively, the BTF image associated with the corresponding illumination and view directions can be reconstructed by performing $\mathcal{C}_{BTF} \times_3 \mathbf{l}^T \times_4 \mathbf{v}^T$. Fig. 4(b) shows reconstructed BTF images of LEGO.

Our tensor representation has clear advantages compared to the one used in [Vasilescu and Terzopoulos 2004]. First, by representing an image as a second-order subtensor, the redundancies between adjacent rows and columns can be removed and much higher compression ratios can be achieved. Second, the resulting basis matrices $\mathbf{U}^{row}$ and $\mathbf{U}^{col}$ in (4) are much smaller than the matrix $\mathbf{U}_{texel}$ in [Vasilescu and Terzopoulos 2004], and therefore much less expensive to compute. Third, because there are more basis matrices, each of them does not need to be truncated as much to achieve the same compression ratio.



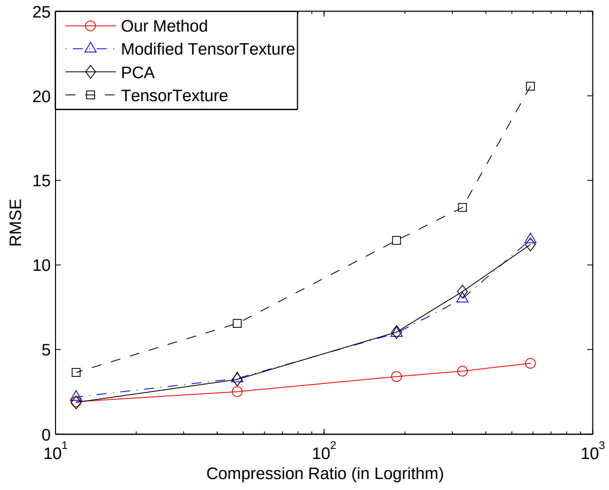
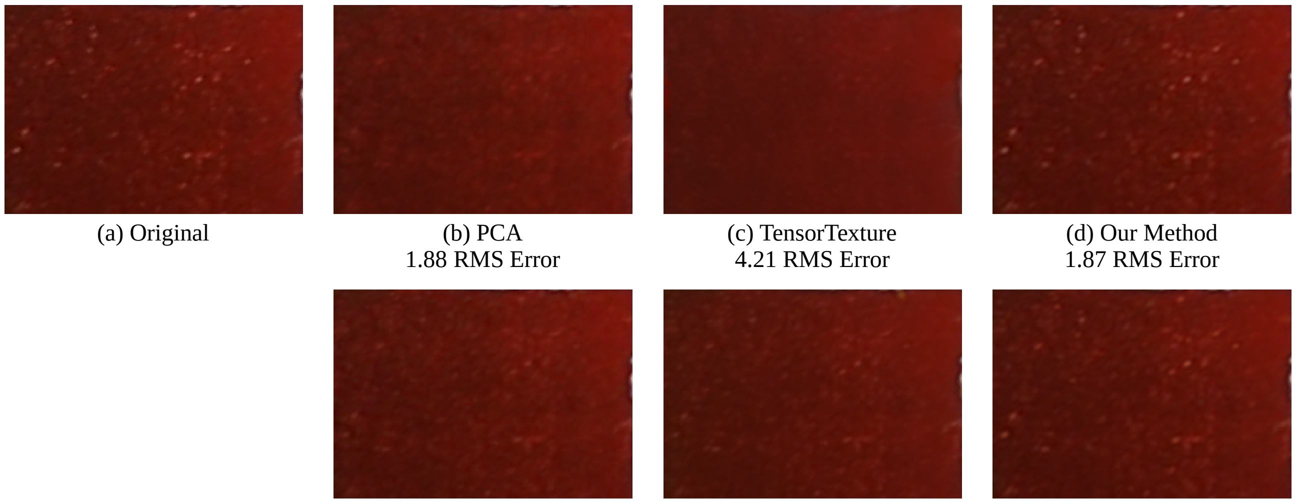


Figure 5: Comparisons of RMS errors obtained for different compression ratios by PCA (black diamonds), TensorTexture (black squares), modified TensorTexture (blue triangles), and our method (red circles). The top diagram shows the errors for a LICHEN BTF, and the bottom one shows the errors for a VELVET BTF.

Fig. 6 and 7 compare our tensor representation against PCA and TensorTexture [Vasilescu and Terzopoulos 2004] using the captured BTFs from [Koudelka et al. 2003]. For the convenience of comparison, we actually used 45 views and 60 illumination directions from each BTF, and the image resolution is 192x192. It can be easily verified that processing each color channel separately does not affect tensor approximation results. For each channel, our scheme constructs a 4th-order tensor with size 192x192x45x60; TensorTexture adopts a 3rd-order tensor with size 36864x45x60; PCA organizes the data into a matrix of size 36864x2700. Note that TensorTexture only compresses the view and illumination modes while maintaining the original image resolution. We also did experiments on a



(a) Original

(b) PCA
1.88 RMS Error

(c) TensorTexture
4.21 RMS Error

(d) Our Method
1.87 RMS Error

(e) Modified TensorTexture
2.19 RMS Error

(f) TensorTexture
5.71 RMS Error

(g) Our Method
1.92 RMS Error

Figure 6: A comparison of reconstructed images from PCA, TensorTexture and our tensor representation. The original VELVET BTF has 45 views and 60 illumination directions, and the image resolution is 192x192. A compression ratio of 12, which is equivalent to 91.7% compression, is used for all the methods. (a) Original image. (b) PCA. (c)&(f) TensorTexture. There are 15 views and 15 illums for (c), and 25 views and 9 illums for (f). TensorTexture does not perform spatial compression. (e) Modified TensorTexture with spatial compression. There are 30 views and 30 illums, but the spatial resolution is compressed to 96x96. (d)&(g) Our method. There are 30 views and 30 illums for (d), and 45 views and 20 illums for (g). The spatial resolution is compressed to 96x96 in both.

modified version of TensorTexture, which compresses the spatial resolution as well. The only difference between our scheme and modified TensorTexture is that our scheme considers an image as a 2nd-order subtensor while modified TensorTexture considers an image as a vector.

Fig. 5 shows comparisons of RMS errors obtained by the aforementioned methods for different compression ratios. Our method consistently produces the least RMS error in all scenarios. It should be noted that PCA produces comparable results for the VELVET sample at small compression ratios. The RMS errors for our method were obtained using the same reduced ranks for both view and illumination. The results can be further improved if we tune the ranks of these two modes separately. The RMS errors for TensorTexture were obtained by choosing the minimum error among three cases where the rank for view is greater than, equal, or less than the rank for illumination, respectively. The RMS error is actually not a good error metric for the VELVET BTF shown in Fig. 6 since the velvet surface has tiny light-colored dots which tend not to affect RMS significantly. Nevertheless, these dots are preserved surprisingly well by our method.

It can be easily verified that the reconstructed images from our tensor representation produces the best results both visually and numerically among the methods we have tested. Visually, our method faithfully preserves high-resolution details during compression. In terms of RMS error, our scheme is the only one that generates an error lower than PCA. The same conclusions hold for all the BTFs we have tested. In the worst case, our result is still slightly better than PCA. In general, as long as there are a relatively large number of image rows and columns and there is redundancy among them, our subtensor representation can achieve a significant improvement because only a small set of basis vectors are necessary to approximate all the rows and columns. We did not use our out-of-core technique (Section 3.3) on these examples because they are sufficiently small. However, this has no bearing on the results obtained as our out-of-core algorithm only manages memory usage and does not affect the quality of tensor approximation.

In a separate set of experiments, our out-of-core algorithm has been successfully applied to the complete BTFs in [Koudelka et al. 2003]. These BTFs have 90 view directions and 120 illumination directions. Our out-of-core algorithm performed significantly better than the PCA-based compression step in [Koudelka et al. 2003]. For the VELVET BTF, the RMS error of the PCA-based compression in [Koudelka et al. 2003] is 4.02 if 150 basis vectors are used while the RMS error of our tensor-based algorithm is only 2.24 if the same compression ratio is achieved. Just as the PCA coefficients can be encoded with JPEG [Koudelka et al. 2003], the coefficients in our core tensor $\mathcal{B}_{vol}$ can also be further encoded to boost the compression ratio. However, this encoding step is left as our future work.

Fig. 1 shows a synthetic image of a scene with two BTF-mapped objects. The BTFs are represented as compressed tensors.

4.2 Dynamic BTF modeling and compression

A conventional BTF describes the appearance of a static surface region while there are many deforming surfaces. We define a dynamic BTF by associating a potentially distinct BTF with any instant in time. Such a dynamic BTF describes the dynamic appearance of a surface with changing geometric or photometric properties. With the additional temporal axis, a dynamic BTF is a seven-dimensional function that demands a compact representation. There has been previous work on rendering deformable objects using a static BTF [Furukawa et al. 2002] or reflectance field [Weyrich et al. 2005]. The basic idea is to resample the static BTF according to the deformation. Our dynamic BTF is different in that the BTFs associated with different times are not simply the warped versions of a static BTF. They are true changing BTFs of the underlying dynamic surface.

We organize a dynamic BTF as a fifth-order tensor, $\mathcal{A}_{DBTF} \in \Re^{I_{row} \times I_{col} \times I_{illum} \times I_{view} \times I_{time}}$, using the same notation as in Section 4.1 and I_{time} is the number of time samples (frames). By running the out-of-core algorithm in Section 3.3, we can obtain a new tensor $\tilde{\mathcal{A}}_{DBTF}$ which best approximates $\mathcal{A}_{DBTF}$. The new

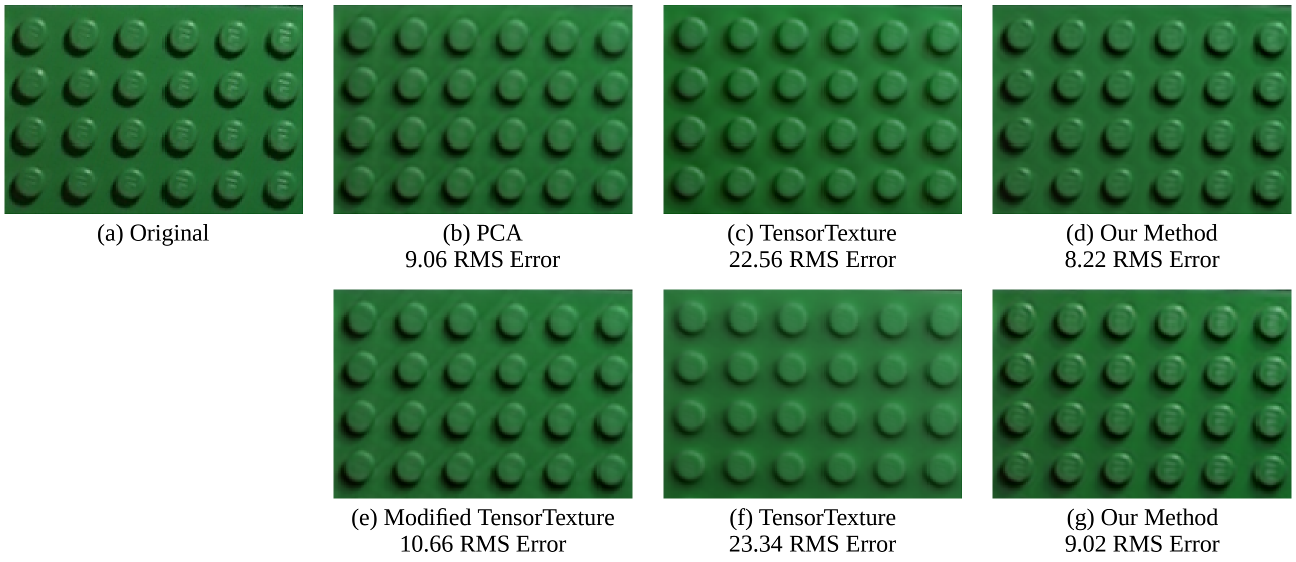


Figure 7: A comparison of reconstructed images from PCA, TensorTexture and our tensor representation. The original LEGO BTF has 45 views and 60 illumination directions, and the image resolution is 192x192. A compression ratio of 48, which is equivalent to 97.9% compression, is used for all the methods. (a) Original image. (b) PCA. (c)&(f) TensorTexture. There are 7 views and 8 illums for (c), and 14 views and 4 illums for (f). TensorTexture does not perform spatial compression. (e) Modified TensorTexture with spatial compression. There are 30 views and 30 illums, but the spatial resolution is compressed to 48x48. (d)&(g) Our method. There are 30 views and 30 illums for (d), and 45 views and 20 illums for (g). The spatial resolution is compressed to 48x48 in both.

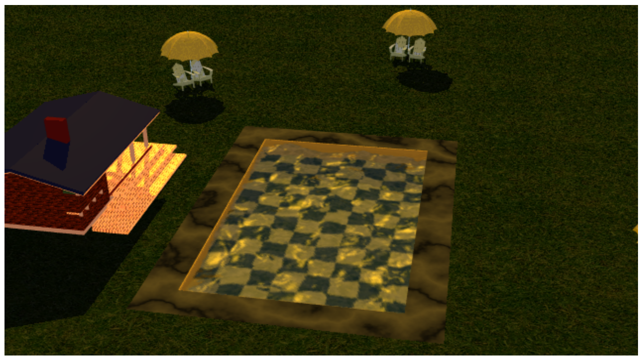


Figure 8: A water pool mapped with a dynamic BTF in tensor representation. This image shows the appearance of the pool during sunset.

tensor is formulated as:

$$\tilde{\mathcal{A}}_{DBTF} = \mathcal{B}_{DBTF} \times_1 \mathbf{U}^{row} \times_2 \mathbf{U}^{col} \times_3 \mathbf{U}^{illum} \times_4 \mathbf{U}^{view} \times_5 \mathbf{U}^{time}, \tag{5}$$

where $\mathbf{U}^{time} \in \Re^{I_{time} \times R_{time}}$, and R_{time} is the reduced rank for the temporal mode.

We have applied our out-of-core tensor approximation to a synthetically generated dynamic BTF which specifies the appearances of the dynamic water surface in a pool. 240 frames of the dynamic water surface were first generated from a computational fluid simulator based on [Enright et al. 2002]. Then for the water surface in each simulated frame, we use a ray tracer to synthetically render a set of BTF images for all combinations of 28 views and 28 illumination directions. The maximum resolution of the image region covering the pool is 180x300. It took nine 2.4GHz PCs more than one week to precompute all the images. The total amount of data is about 30GB. The size of the initial 5th-order tensor for each color channel of this dynamic BTF is 180x300x28x28x240. Our out-of-core algorithm divided each of these huge tensors into 72 blocks and successfully compressed it down to 75x150x28x14x80. It took a 3.4GHz Intel processor with 2GB memory around 60 hours to finish the computation.

Once we have this compressed tensor, it takes only 2 seconds to reconstruct an image for the pool given an arbitrary time and an arbitrary pair of view and illumination directions. This performance is achieved by software rendering on the same Intel 3.4GHz processor. It can be potentially improved by GPU-based rendering which is left for future research. Given the compressed tensor with all the information, we can extract interesting subtensors. If we fix the illumination and view directions and vary time, we are simply playing back a conventional fluid animation. If we fix the time only, we obtain a BTF for the water surface at a specific instant. If we fix the view only, we can play back the fluid animation with a simultaneously changing lighting condition. Fig. 8 shows one frame of an animation generated in this way. Complete animations generated from the compressed tensor can be found in the accompanying video. Note that the interpolation scheme for the view and illumination directions [Vasilescu and Terzopoulos 2004] can be applied to time as well. Thus, we can play back different portions of the water motion at different speeds. Animations with varying speeds can also be found in the video.

4.3 Temporal volume data compression

Many computer animation and simulation techniques generate data in a three-dimensional volume grid. The generated volume data is actually four-dimensional with the volume grid accounting for three dimensions and the time being the fourth. The size of such datasets is enormous, and needs compression, especially when a grid resolution is high. Although much work has been performed on compressing static 3D volume data using mathematical tools, such as DCT and wavelet decomposition [Yeo and Liu 1995; Rodler 1999; Nguyen and Saupe 2001], techniques for compressing 4D dynamic volume data as a whole are still under development.

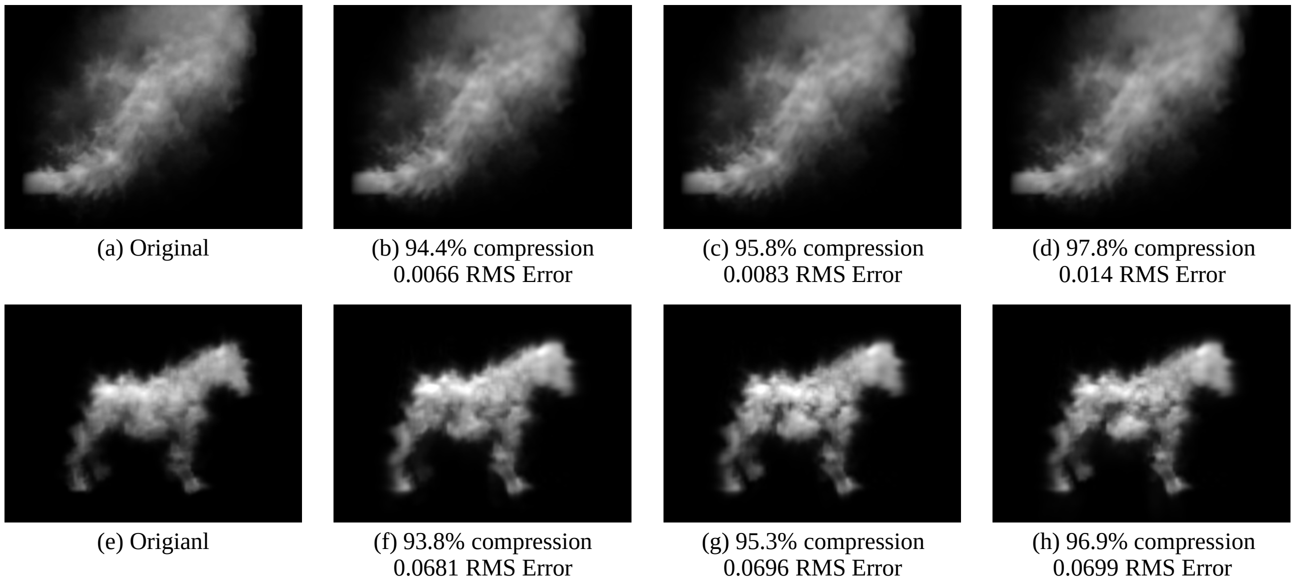


Figure 9: Tensor approximation of volumetric smoke simulations. (a) An original smoke simulation with 480 temporal frames and a volume resolution of 128x128x128. The smoke density values fall between the interval, [0, 1.065]. The spatial resolution is compressed to 64x64x64 in (b)-(d). The temporal resolution is compressed to 200 in (b), 150 in (c) and 80 in (d). Each out-of-core compression divided the original tensor into 32 blocks and took 11.3 hours on a 2.4GHz processor with 784MB RAM. (e) Another smoke simulation with 320 temporal frames and a volume resolution of 100x100x100. The smoke density values fall between the interval, [0, 5.688]. The spatial resolution is compressed to 50x50x50 in (f)-(h). The temporal resolution is compressed to 160 in (f), 120 in (g) and 80 in (h). Each out-of-core compression divided the original tensor into 32 blocks and took 2.2 hours on a 2.4GHz processor with 784MB RAM.

We apply tensor-based approximation again to dynamic volume data to obtain a compressed representation. Initially, the original data is organized as a fourth-order tensor, $\mathcal{A}_{vol} \in \Re^{I_X \times I_Y \times I_Z \times I_{time}}$, where I_X, I_Y and I_Z represent the grid resolution along each dimension of the volume, and I_{time} is the number of simulated frames. To remove redundancy, we use another fourth-order tenser $\tilde{\mathcal{A}}_{vol}$ with reduced ranks to approximate $\mathcal{A}_{vol}$ as in Section 2.2. The new tensor is represented as:

$$\tilde{\mathcal{A}}_{vol} = \mathcal{B}_{vol} \times_1 \mathbf{U}^X \times_2 \mathbf{U}^Y \times_3 \mathbf{U}^Z \times_4 \mathbf{U}^{time}, \qquad (6)$$

where $\mathbf{U}^X \in \Re^{I_X \times R_X}$, $\mathbf{U}^Y \in \Re^{I_Y \times R_Y}$, $\mathbf{U}^Z \in \Re^{I_Z \times R_Z}$, $\mathbf{U}^{time} \in \Re^{I_{time} \times R_{time}}$, and $\mathcal{B}_{vol} \in \Re^{R_X \times R_Y \times R_Z \times R_{time}}$. R_X, R_Y, R_Z and R_{time} specify the reduced ranks (columns) of the four basis matrices for the four modes. Without decomposing the three dimensions of a volume using three basis matrices as suggested in Section 3.1, we would have to convert each volume into an extremely long vector, which is infeasible.

We have successfully applied our out-of-core tensor approximation to multiple smoke simulation sequences each of which has between 2.5GB and 10GB of original data. They were originally generated from a controllable smoke simulator in [Shi and Yu 2005]. Our out-of-core algorithm was run on a PC with only 784MB memory. In each simulated frame, there is a 3D smoke density distribution of the uniform volume grid. We store such density distributions on disks using double-precision floating point numbers. They typically have smooth spatial variations and strong temporal coherence, creating much spatial and temporal redundancy. Results for two smoke sequences are shown in Fig. 9 where the images are synthetically rendered using smoke density distributions obtained after tensor-based compression and decompression. These images are quite similar to the images rendered from the original smoke density distributions.

5 Discussions

We presented an out-of-core algorithm for higher-order tensor approximation. The basic idea is to partition a tensor into smaller blocks and perform tensor-related operations blockwise. An additional feature of this algorithm is that it effectively exploits the spatial redundancy of multi-dimensional data item to achieve higher compression ratios.

With the increasing popularity of data-driven models in graphics, such an out-of-core algorithm becomes indispensable for extracting compact representations. Although we presented three applications here, the promise of this algorithm extends far beyond that. In future, we would like to consider additional applications, such as light fields, complete reflectance fields and other physically simulated data. It is also possible to use this algorithm for data with irregular connectivity, such as meshes, by parameterizing them onto a rectangular domain [Gu et al. 2002]. We would like to explore the prospects of using GPUs to perform tensor reconstruction and multilinear image rendering. We also plan to further design an efficient encoder based on tensor approximation for multi-dimensional datasets. Currently, the reduced ranks of the approximating tensor are specified by the user. It would be helpful if they can be automatically and efficiently found given a desired RMS error threshold.

Acknowledgments

We wish to thank the authors of [Koudelka et al. 2003] for sharing their BTF database, and the anonymous reviewers for their valuable comments. This work was partially supported by National Science Foundation (CCR-0132970) and Office of Naval Research (N00014-03-1-0107). The first author was supported by UIUC CSE fellowship.

References

BRAND, M. 2002. Incremental singular value decomposition of uncertain data with missing values. In *Proc. European Conference on Computer Vision (Vol. I)*, 707–720.

CHEN, W.-C., BOUGUET, J.-Y., CHU, M., AND GRZESZCZUK, R. 2002. Light field mapping: Efficient representation and hardware rendering of surface light fields. *ACM Transactions on Graphics 21*, 3, 447–456.

DANA, K. J., VAN GINNEKEN, B., NAYAR, S. K., AND KOENDERINK, J. J. 1999. Reflectance and texture of real world surfaces. *ACM Transactions on Graphics 18*, 1, 1–34.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Transactions on Graphics 21*, 3, 736–744.

FURUKAWA, R., KAWASAKI, H., IKEUCHI, K., AND SAKAUCHI, M. 2002. Appearance based object modeling using texture database: Acquisition, compression, and rendering. In *13th Eurographics Workshop on Rendering*, 257–265.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry images. *ACM Transactions on Graphics 21*, 3, 355–361.

HAN, J., AND PERLIN, K. 2003. Measuring bidirectional texture reflectance with a kaleidoscope. *ACM Transactions on Graphics 22*, 3, 741–748.

JAMES, D., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM TOG 22*, 3, 879–887.

KOUDELKA, M., MAGDA, S., BELHUMEUR, P., AND KRIEGMAN, D. 2003. Acquisition, compression, and synthesis of bidirectional texture functions. In *3rd Intl. Workshop on Texture Analysis and Synthesis*, 59–64.

KROONENBERG, P., AND DE LEEUW, J. 1980. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika 45*, 324–1342.

LATHAUWER, L. D., DE MOOR, B., AND VANDEWALLE, J. 2000. A multilinear singular value decomposition. *SIAM J. Matrix Analysis and Applications 21*, 4, 1253–1278.

LATHAUWER, L. D., DE MOOR, B., AND VANDEWALLE, J. 2000. On the best rank-1 and rank-($R_1, R_2, ..., R_n$) approximation of higher-order tensors. *SIAM J. Matrix Analysis and Applications 21*, 4, 1324–1342.

LEUNG, T., AND MALIK, J. 1999. Recognizing surfaces using three dimensional textons. In *Intl. Conf. Computer Vision*.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. In *Computer Graphics Proceedings, Annual Conference Series*, 31–42.

LIU, X., YU, Y., AND SHUM, H.-Y. 2001. Synthesizing bidirectional texture functions for real-world surfaces. In *Proceedings of SIGGRAPH*, 97–106.

LIU, X., HU, Y., ZHANG, J., TONG, X., GUO, B., AND SHUM, H.-Y. 2004. Synthesis and rendering of bidirectional texture functions on arbitrary surfaces. *IEEE Trans. Visualization and Computer Graphics 10*, 3, 278–289.

MATUSIK, W., PFISTER, H., BRAND, M., AND MCMILLAN, L. 2003. A data-driven reflectance model. *ACM Transactions on Graphics 22*, 3, 759–769.

NGUYEN, K., AND SAUPE, D. 2001. Rapid high quality compression of volume data for visualization. *Compuer Graphics Forum 20*, 3, 49–56.

NISHINO, K., SATO, Y., AND IKEUCHI, K. 1999. Eigen-texture method: appearance compression based on 3d model. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR'99)*, 618–624.

RABANI, E., AND TOLEDO, S. 2001. Out-of-core svd and qr decompositions. In *Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing*.

RODLER, F. 1999. Wavelet based 3d compression with fast random access for very large volume data. In *Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, 108–117.

SATTLER, M., SARLETTE, R., AND KLEIN, R. 2003. Efficient and realistic visualization of cloth. In *Proc. Eurographics Symposium on Rendering*, 167–177.

SHASHUA, A., AND LEVIN, A. 2001. Linear image regression and classification using the tensor-rank principle. In *IEEE Conf. Computer Vision and Pattern Recognition*.

SHI, L., AND YU, Y. 2005. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics 24*, 1, 140–164.

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. In *SIGGRAPH 2002 Proceedings*, 665–672.

TUCKER, L. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika 31*, 279–311.

VASILESCU, M. A. O., AND TERZOPOULOS, D. 2002. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, 447–460.

VASILESCU, M., AND TERZOPOULOS, D. 2004. Tensortextures: Multilinear image-based rendering. *ACM Transactions on Graphics 23*, 3, 334–340.

WANG, H., AND AHUJA, N. 2003. Facial expression decomposition. In *Int. Conf. on Computer Vision*, 958–965.

WEYRICH, T., PFISTER, H., AND GROSS, M. 2005. Rendering deformable surface reflectance fields. *IEEE Trans. Visualization and Computer Graphics 11*, 1, 48–58.

YANG, J., ZHANG, D., FRANGI, A., AND YANG, J. 2004. Two-dimensional pca: A new approach to appearance-based face representation and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 1 (January).

YE, J. 2004. Generalized low rank approximations of matrices. In *International Conference on Machine Learning, ICML'04*.

YEO, B.-L., AND LIU, B. 1995. Volume rendering of dct-based compressed 3d scalar data. *IEEE Trans. Visualization and Computer Graphics 1*, 1, 29–43.

YU, Y., AND CHANG, J. 2005. Shadow graphs and 3d texture reconstruction. *International Journal of Computer Vision 62*, 1/2, 35–60.