Eurographics Symposium on Rendering (2007)
Jan Kautz and Sumanta Pattanaik (Editors)

Large-Scale Data Management for PRT-Based Real-Time Rendering of Dynamically Skinned Models

Wei-Wen Feng[1] Liang Peng[2] Yuntao Jia[1] Yizhou Yu[1]

[1]University of Illinois at Urbana-Champaign
[2]Rambus Inc.

Abstract

Computer games and real-time applications frequently adopt mesh skinning as a deformation technique for virtual characters and articulated objects. Rendering skinned models with global shading effects, such as interreflection and subsurface scattering, using precomputed radiance transfer enables high-quality real-time display of dynamically deformed objects. In this approach, we need to precompute radiance transfer for many sampled poses. Resulting datasets reach hundreds of gigabytes, and are orders of magnitude larger than those for a static object. This paper presents simple but effective large-scale data management techniques so that runtime data communication, decompression and interpolation can be performed efficiently and accurately. Specifically, we have developed a mesh clustering technique based on spectral graph partitioning to facilitate interpolation from nearest neighbors and an incremental clustering method for transfer matrix compression. By exploiting additional data redundancies among different sampled poses, we can achieve higher compression ratios with the same fidelity. Our incremental clustering can make the runtime cost of per-frame data decompression and interpolation satisfy a prescribed upper bound. As a result, we can achieve real-time performance using the massive precomputed data and an efficient runtime algorithm.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-dimensional Graphics and Realismcolor, shading, shadowing, and texture

1. Introduction

Precomputed radiance transfer (PRT) provides the opportunity to produce compelling realism with global shading effects in real time. Originally developed for static scenes in low-frequency [SKS02, SHHS03] or all-frequency settings [LSSS04, NRH04, WTL04, TS06], PRT has been subsequently extended to fixed animation sequences as well as deformable objects with shading effects caused by detailed surface features but without cast shadows [SLS05]. It has also inspired techniques that produce soft shadows for dynamic scenes [ZHL*05, RWS*06] as well as algorithms for real-time lighting design [KAMJ05] and cinematic relighting [HPB06]. However, the generalization of PRT to dynamic scenes with global shading effects, such as interreflection and subsurface scattering, has not been very successful. Our goal is to overcome this limitation on the class of deformable objects generated by skinning [LCF00, MG03], which is the most widely adopted deformation technique for virtual characters and articulated objects in computer games and real-time applications.

In skinning, there is a set of rigid "bones". Surface deformations are defined as functions of the rigid movements of nearby bones [LCF00,MG03,JT05]. Such a nature gives rise to deformations at two different scales. The relative position among large surface segments, such as the limbs of a virtual character, may undergo large-scale changes. Nevertheless, large-scale movements are highly correlated. For example, a virtual character is typically designated with a few classes of whole body movements, such as walking, running and dancing. Each class of movements exhibit a high degree of coordination among various body parts [SHP04]. Subspaces that enclose relevant poses can be identified with a covariance analysis on sample movements. At a smaller scale, deformations within each surface region, such as the bulging of muscles, are smoothly varying, and spatially close points share similar deformations.

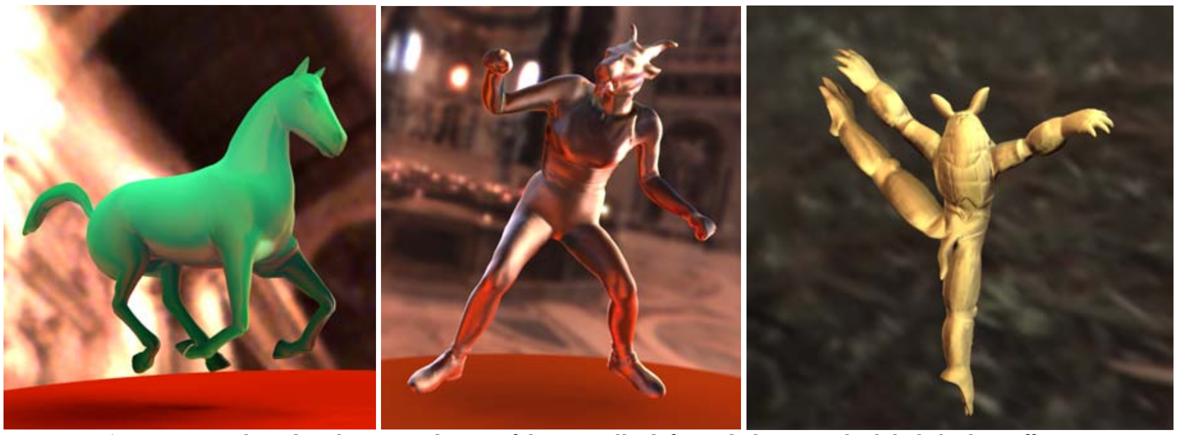


Figure 1: *PRT-based real-time rendering of dynamically deformed objects with global shading effects.*

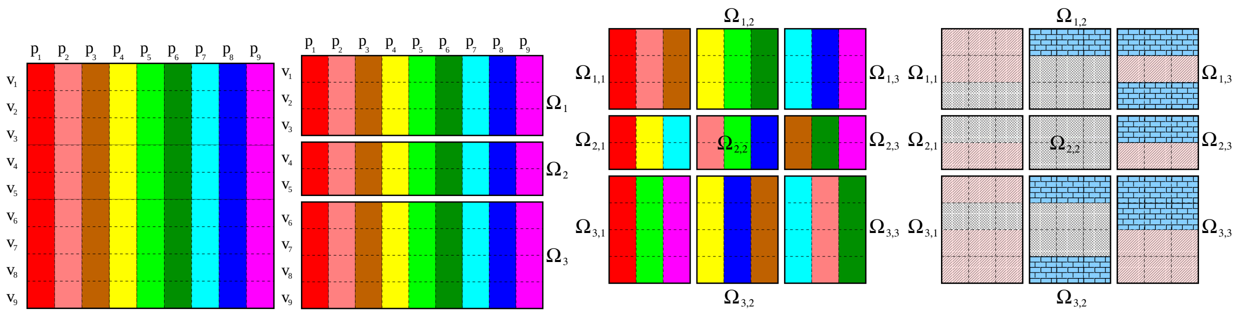
In this paper, we focus on real-time PRT techniques that can achieve realistic global shading effects, such as interreflection and subsurface scattering, on glossy or translucent surfaces deformed by skinning. Most importantly, such surface deformation is not precomputed, but dynamically generated. To be able to produce these global shading effects, we take the example-based approach that draws many samples in the pose subspaces for a particular object and precomputes radiance transfer for them. Note that during runtime a dynamically generated pose as well as its associated surface deformation may be different from all the sampled poses. Therefore, some type of interpolation from nearest neighbors in the pose space is inevitable.

An important question is what criteria or distance metrics we should use to search for these nearest neighbors. Pose similarity should obviously be considered. But pose similarity alone is insufficient. Even a subspace of poses typically has multiple dimensions. Any practical number of sampled poses only give rise to a sparse sampling within such a space, which results in large interpolation errors. On the other hand, the configuration of a single surface segment lies in a much lower dimensional space. Therefore, we search for nearest neighbors for each surface segment separately and then perform interpolation using different nearest neighbors for different surface segments. When searching for such segment-wise nearest neighbors, we also consider similarity in terms of local PRT data (i.e. the precomputed radiance transfer matrices). Unlike local geometry, radiance transfer matrices over a surface segment actually account for global effects such as scattering and interreflections caused by other surface segments.

Another tough challenge we face is the sheer size of the data generated by precomputing. Because we need to precompute radiance transfer for every sampled pose, resulting datasets reach hundreds of gigabytes, and are orders of magnitude larger than those for a static object. We need clustering and compression methods that are better suited for such large-scale data for the following reasons. First, compressed datasets need to fit into the system memory. Otherwise, dynamically loading data from hard disks during runtime would be intolerably slow. Second, because loading all compressed data into the GPU memory has become infeasible, effective data compression can reduce per-frame communication overhead between the CPU and GPU. Furthermore, these goals should be achieved without compromising rendering quality or increasing per-frame data decompression cost, which is directly related to the frame rate we can achieve.

In this paper, we present effective clustering and compression schemes for precomputed radiance transfer matrices so that the aforementioned runtime data communication, decompression and interpolation can be performed efficiently and accurately. First, we have developed a data segmentation scheme to facilitate interpolation from nearest neighbors. Once all the meshes associated with the sampled poses have been consistently segmented in the surface domain, every group of corresponding surface segments are further evenly divided into small groups in the pose domain using spectral graph partitioning and a nonlinear measure that computes similarities in terms of both pose and radiance transfer matrices. Second, we have developed a revised clustered PCA algorithm for transfer matrices. It incrementally creates new PCA clusters for data associated with new pose configurations. By exploiting additional data redundancies among different sampled poses, our method can achieve a higher compression ratio with the same approximation error. Our incremental clustering can make the cost of per-frame data decompression and interpolation satisfy a prescribed upper bound.

In addition, we have designed an efficient runtime algorithm that takes advantage of the precomputed clustering and compression results while effectively distributing the workload among multiple rendering passes. As a result, high-



(a) Input (b) Consistent Segmentation (c) Mesh Segment Clustering (d) Revised Clustered PCA

Figure 2: *A multistage pipeline for data segmentation and clustering in the joint spatio-pose space. (a) PRT data for all combinations of poses and vertices can be arranged into a large-scale matrix. (b) Consistent segmentation horizontally divides the matrix into multiple submatrices. (c) Mesh segment clustering reorganizes columns of each submatrix into multiple smaller clusters. (d) Revised clustered PCA projects each row of the clusters onto the basis vectors of the same PCA cluster to facilitate runtime interpolation. Vertices in the same PCA cluster are shown with the same pattern.*

quality real-time rendering with global shading effects is achieved.

2. Overview

Since a dynamic articulated object may have a number of linked parts moving simultaneously, interactively computing light transport, including glossy reflection, interreflection, shadowing and subsurface scattering, under environment illumination for such objects is extremely challenging. Therefore, state-of-the-art techniques can only dynamically generate soft shadows [RWS*06], all-frequency shadows [SM06], or ambient occlusion values [KA06a, KA06b, KA07]. There exist other limitations with these techniques. In fact, the algorithm in [SM06] can interactively generate shadows for one moving part only, and only diffuse shading has been demonstrated on original object surfaces in [RWS*06].

To achieve all global shading effects on glossy or translucent articulated objects, we take the approach that draws many samples in the pose subspaces for a particular object, precomputes radiance transfer for all of them, and interpolates the precomputed transfer matrices during runtime. Note that the high-level approach of the rendering stage in [JF03] is similar to ours. Nevertheless, they only devoloped methods for diffuse surfaces and small elasticity-based deformations while our articulated objects typically generate deformations at a much larger scale. The data driven approach adopted by our paper is also used for real-time ambient occlusion in [KA06b]. In their work, a linear model for predicting ambient occlusion is derived from a set of training data with various poses. The ambient occlusion is a one dimensional scalar value representing the proportion of occlusion while the dimension of the transfer matrix is much higher, and thus pose a bigger challenge for both data management and run-time performance. In this paper, we adopt the basic PRT framework for glossy objects presented in [SKS02, SHHS03] which account for global shading effects caused by low-frequency environment lighting using a truncated spherical harmonic basis.

Since we need to sample poses and perform interpolation, we briefly discuss our pose sampling strategy here even though it is not the focus of this paper. We took multiple representative sequences from the CMU motion capture database [MoC]. Each sequence represents a distinct type of whole body motion, such as boxing, dancing, running and walking. A pose consists of all the joint angles in a specific skeletal configuration of the object, and there is a pose at every frame of these sequences. We ran clustered principal component analysis (CPCA) on all the poses and extracted a few pose subspace clusters. For each cluster generated this way, we choose the first eight principle components representing the variations of pose subspace within that cluster. We can adjust these eight PCA coefficients to generate new poses. Since we would like to obtain pose samples that are distributed more uniformly and widely within each subspace than the input poses, we resample each subspace by performing stratified Monte Carlo sampling on the PCA coefficients. A set of sampled PCA coefficients along with the principle components are used for generating a sampled pose. Let $\mathcal{C} = \{p_1, p_2, ..., p_{m_p}\}$ be the set of resampled poses. Note that we need to generate a deformed surface mesh for every sampled pose using skinning and then precompute radiance transfer matrices for these deformed meshes. All the deformed meshes have the same number of vertices and the same connectivity among the vertices. Only the vertex positions have been altered. Let $\Upsilon^j = \{v_1^j, v_2^j, ..., v_{n_v}^j\}$ be the complete set of vertices in the deformed mesh for a single pose $p_j \in \mathcal{C}$. We can arrange all the vertices within Υ^j into a single column vector, and arrange such column vectors for all poses into a matrix, $\mathcal{M}$. Each row of vertices within $\mathcal{M}$ are actually corresponding vertices on different meshes (Fig. 2(a)).

We perform a consistent mesh segmentation across all the deformed meshes using the technique in [JT05] to obtain a set of segments for every mesh. A consistent segmentation partitions different meshes into the same number of segments and corresponding vertices on different meshes belong to corresponding segments (Fig. 2(b)). Let $\{\Upsilon_k^j | 1 \le k \le n_r\}$ be the segmentation of Υ^j so that $\Upsilon^j = \bigcup_k \Upsilon_k^j$. Next, we group corresponding segments on different meshes together and define $\Omega_k = \{\Upsilon_k^j | 1 \le j \le M\}$ for $1 \le k \le n_r$. We can imagine that each Ω_k is actually a submatrix of $\mathcal{M}$, and occupies a subset of rows within $\mathcal{M}$. We further run spectral graph partitioning (Section 3) on each Ω_k independently to reorganize its columns and evenly distribute them into smaller clusters each of which typically has 3-8 segments. Let $\{\Omega_{k,g} | 1 \le g \le n_c\}$ be the clusters from dividing Ω_k so that $\Omega_k = \bigcup_g \Omega_{k,g}$. If we move the segments of each cluster together, we can imagine each cluster as a submatrix where each column consists of vertices from the same mesh segment and each row consists of corresponding vertices from different segments within the same cluster (Fig. 2(c)). The mesh segments within each cluster should exhibit two types of similarity. If $\Upsilon_k^{j_1}$ and $\Upsilon_k^{j_2}$ belong to the same cluster, their corresponding poses, p_{j_1} and p_{j_2}, should be similar, and radiance transfer matrices at corresponding vertices of $\Upsilon_k^{j_1}$ and $\Upsilon_k^{j_2}$ should also be similar. After data segmentation, radiance transfer matrices over all sampled poses are finally compressed using a revised clustered PCA algorithm.

During runtime, given an input skeletal configuration as well as the associated surface mesh, which shares the same segmentation discussed above, we first use the skeleton to find the most similar pose among the samples. Suppose this most similar pose is p_j, which is then used to find the cluster Ω_{k,g_k^j} where each mesh segment, $\Upsilon_k^j (1 \le k \le n_r)$, belongs. Radiance transfer matrices at each row of corresponding vertices within each retrieved cluster of mesh segments are finally interpolated to generate estimated transfer matrices for the vertices on the input mesh. There are two major reasons to perform the aforementioned clustering on each group of segments $\Omega_k (1 \le k \le n_r)$. First, the resulting clusters can accelerate runtime nearest neighbor search. We directly consider the mesh segments in the same cluster as approximate nearest neighbors. Second, such clustering can accelerate runtime transfer matrix interpolation once transfer matrices at corresponding mesh vertices in the same cluster of mesh segments are compressed using the same set of PCA basis vectors (Section 4 and Fig. 2(d)).

3. Mesh Segment Clustering

In this section, we apply a spectral graph partitioning algorithm, called *normalized cut* [SM00], to dividing each $\Omega_k (1 \le k \le n_r)$ into smaller clusters of mesh segments. In the current context, clustering based on normalized cut has important advantages. First, unlike linear subspace methods, such as PCA and clustered PCA, normalized cut is based on local pairwise similarity. Therefore, it is better suited for interpolation based on nearest neighbors. Second, normalized cut can easily handle nonlinear similarity measures and can easily integrate multiple similarity measures together to achieve a tradeoff among them. Third, a similarity measure in normalized cut is typically defined as a Gaussian, which is a widely used radial basis function. Such a nonlinear measure is consistent with pose-based skin deformation techniques that use radial basis functions [LCF00]. The normalized cut algorithm has recently been successfully applied to the compression of motion capture sequences [Ari06]. In the following section, we briefly introduce this algorithm first.

3.1. Normalized Cut Framework

Let $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ be a weighted graph, where the set of nodes, $\mathcal{U} = \{u_1, u_2, ..., u_n\}$. An edge, $(u_i, u_j) \in \mathcal{E}$, has a weight $w(u_i, u_j)$ defined by the similarity between the location and attributes of the two nodes defining the edge. The idea is to partition the nodes into two subsets, $\mathcal{A}$ and $\mathcal{B}$, such that the following disassociation measure, the normalized cut, is minimized.

$$Ncut(\mathcal{A},\mathcal{B}) = \frac{cut(\mathcal{A},\mathcal{B})}{asso(\mathcal{A},\mathcal{U})} + \frac{cut(\mathcal{A},\mathcal{B})}{asso(\mathcal{B},\mathcal{U})} \qquad (1)$$

where $cut(\mathcal{A},\mathcal{B}) = \sum_{s\in\mathcal{A},t\in\mathcal{B}} w(s,t)$ is the total connection from nodes in $\mathcal{A}$ to nodes in $\mathcal{B}$; $asso(\mathcal{A},\mathcal{U}) = \sum_{s\in\mathcal{A},t\in\mathcal{U}} w(s,t)$ is the total connection from nodes in $\mathcal{A}$ to all nodes in the graph; and $asso(\mathcal{B},\mathcal{U})$ is similarly defined. This measure works much better than $cut(\mathcal{A},\mathcal{B})$ because it favors relatively balanced subregions instead of cutting small sets of isolated nodes in the graph.

To compute the optimal partition based on the above measure is NP-hard. However, it has been shown [SM00] that a good approximation can be obtained by relaxing the discrete version of the problem to a continuous one which can be solved using eigendecomposition techniques. Let $\mathbf{y}$ be the indicator vector of a partition. Each element of $\mathbf{y}$ takes two discrete values to indicate whether a particular node in the graph belongs to $\mathcal{A}$ or $\mathcal{B}$. If $\mathbf{y}$ is relaxed to take on continuous real values, it can be shown that the optimal solution can be obtained by solving the following generalized eigenvalue system,

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y} \qquad (2)$$

where $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}(i,i) = \sum_j w(u_i,u_j)$, W is the weight matrix with $\mathbf{W}(i,j) = w(u_i,u_j)$. The eigenvector corresponding to the second smallest eigenvalue is the optimal indicator vector in real space. A suboptimal partition can be obtained by first allowing $\mathbf{y}$ to take on continuous real values, solving the above generalized eigenvalue system for $\mathbf{y}$, and then searching for the best threshold to partition the real-valued elements of $\mathbf{y}$ into two subgroups. There exist different criteria to guide this threshold search. By default, one would like to use the cost function in (1) as the criterion so that the threshold can minimize this cost. Alternatively, as in our case, one may look for two evenly subdivided subgroups. Then, the threshold should be the median of the elements in $\mathbf{y}$. In either case, it is a one-dimensional search that can be performed very quickly. The two resulting subregions from this partition can be recursively considered for further subdivision. This algorithm can be used to solve different clustering or segmentation problems by choosing different edge weights [MBLS01].

3.2. Mesh Segment Clustering

In the current context, we would like to partition every group of corresponding mesh segments, $\Omega_k (1 \leq k \leq n_r)$ into multiple smaller clusters. We set up a complete graph for a given group of segments. A node in the graph corresponds to a mesh segment from the group. The attributes of a node include the complete skeletal configuration corresponding to that node as well as the radiance transfer matrices at all the vertices within that segment.

The weight $w(u_i,u_j)$ over an edge (u_i,u_j) is the product of two similarity terms. One measures the overall similarity between the transfer matrices associated with the two nodes of the edge. The other measures the similarity between the poses associated with the two nodes. Both similarity terms can be in the form of a Gaussian distribution. Overall, $w(u_i,u_j)$ is a local measure of how likely the nodes belong to the same partition. $w(u_i,u_j)$ is close to 1 for nodes which are likely to belong together, and close to 0 for nodes which are likely to be separated, as judged purely from local evidence available at the two nodes.

Overall similarity regarding both transfer matrices and pose configuration is formulated as

$$w(u_i,u_j) = \exp\left(-\frac{\sum_l \|\mathbf{T}_l^i - \mathbf{T}_l^j\|_F^2}{2\sigma_t^2} - \frac{\sum_{b\in S} \|\mathbf{R}_b^i - \mathbf{R}_b^j\|_F^2}{2\sigma_p^2}\right) \tag{3}$$

where u_i and u_j are two graph nodes each of which has a corresponding mesh segment, $\mathbf{T}_l^i$ denotes the transfer matrix at vertex v_l in the mesh segment at node u_i, S represents the complete skeleton of an articulated object, b is a “bone" in this skeleton, $\mathbf{R}_b^i$ denotes the local rigid body transform at this “bone" in the pose corresponding to node u_i, and $\mathbf{T}_l^j$ and $\mathbf{R}_b^j$ are defined similarly. We use the Frobenius norm for both transfer matrices and local rigid body transforms. Parameters σ_t and σ_p are automatically determined from the respective standard deviations of the root mean squared differences of corresponding transfer matrices or local transforms within a pair of corresponding mesh segments. These parameters can be further adjusted to reflect the relative weighting between the two similarity measures. In our experiment, we set σ_t^2 equal to 1.0 times the variance of transfer matrices, and σ_p^2 equal to 2.0 times the variance of global poses.

Once the graph is set up, it is recursively partitioned into two subgraphs using the aforementioned normalized cut algorithm. Binary partition only needs one eigenvector with the second smallest eigenvalue with respect to (2). Note that since we favor over the similarity between transfer matrices than poses, the resulting pose clustering will be different than the clustering results based on only the pose differences. In terms of data compression, it is always favorable to exploit the data redundancy. Our clustering method tends to group data with similar transfer matrices together and subsequently improve the compression results.

Since we always use the same number of neighbors to perform runtime interpolation and these neighbors are always from the same cluster, we enforce final clusters from graph subdivision have an equal size. This is achieved by enforcing the two subgraphs from every binary partition have an equal size. Recursive partition terminates when the size of the subgraphs reaches a given threshold, which is typically 4 in our experiments.

4. Transfer Matrix Compression

We need to precompute a radiance transfer matrix at every vertex of the mesh for every pose, which gives rise to an amount of data that is three orders of magnitude larger than that for a static object. However, there exist extra redundancies across different meshes that we can exploit by compressing transfer matrices from different meshes together. Nevertheless, there are two important requirements that need to be satisfied. First, the cost of per-frame data decompression should be bounded to guarantee runtime performance. Second, since runtime interpolation will be performed among transfer matrices at corresponding vertices within the same cluster of mesh segments, they should be compressed in such a way that facilitates such interpolation.

We have developed a revised clustered PCA algorithm with incremental cluster creation to overcome these difficulties. Note that incremental cluster creation is different from incremental singular value decomposition algorithms [Bra02] that incrementally update the basis vectors of a cluster. Clustered PCA divides the original dataset into a few clusters each of which is approximated separately using a truncated PCA basis. In this context, the first requirement translates to an upper bound on the number of distinct PCA clusters used by per-frame data since all our PCA clusters have a fixed number of basis vectors. The second requirement is satisfied by using the basis vectors of the same PCA cluster to approximate all transfer matrices falling on the same row of the same cluster of mesh segments (Fig. 2(d)).

Our incremental algorithm goes through the set of poses in a sequential order and incrementally creates new PCA clusters as necessary for each additional pose while guaranteeing the total number of distinct PCA clusters used by that pose is below a prescribed upper bound, m_{cpf}. Suppose we are looking at pose p_j. Let $E = \{K_i | 1 \leq i \leq m_{pca}\}$ be the set of existing PCA clusters each of which has a list of transfer matrices that have been assigned to it. The mesh for p_j has a set of vertices Υ^j, which has been partitioned into segments, $\{\Upsilon_k^j | 1 \le k \le n_r\}$. Suppose $\Upsilon_k^j \in \Omega_{k,g_k^j}$, where Ω_{k,g_k^j} is a cluster of mesh segments, as defined in Section 2. Since vertices on other mesh segments in Ω_{k,g_k^j} might have been assigned PCA cluster memberships, according to the second requirement, corresponding vertices in Υ_k^j should have the same membership as well. Thus, we can divide $\{\Omega_{k,g_k^j} | 1 \le k \le n_r\}$ into two subgroups, $\{\Omega_{k,g_k^j} | k \in I_a\}$ and $\{\Omega_{k,g_k^j} | k \in I_b\}$, where vertices of the mesh segments in the first subgroup of clusters have been assigned to PCA clusters in E but vertices associated with the second subgroup have not. The second subgroup of clusters are called active clusters. Let $E_j = \{K_i | i \in I_j\} \subseteq E$ be the set of PCA clusters already used by vertices from pose p_j and $\rho^j_{assigned}$ be the percentage of the vertices from p_j that have been assigned PCA clusters. When $1 - \rho^j_{assigned} \le r \frac{m_{cpf} - |E_j|}{m_{cpf}}$, where r is a ratio typically set to 2, we perform an incremental clustering step; otherwise, the number of new PCA clusters that can be created within the per-frame budget is too few compared to the number of unassigned vertices and, therefore, we need to perform a reclustering step to reinitialize a sufficient number of new PCA clusters better suited for the pose under consideration.

4.1. Incremental Clustering

We create $m_{cpf} - |E_j|$ new PCA clusters, and every vertex of the segments in the second subgroup is assigned to either one of the new clusters or one of the existing clusters in E_j. We have revised the original clustered PCA algorithm to achieve this goal. In the revised CPCA algorithm, we assign every row of corresponding vertices within the same cluster of segments to the same PCA cluster. According to the mesh clustering process presented in the previous section, the transfer matrices at such corresponding vertices should already have a high degree of similarity. Therefore, assigning them to the same PCA cluster would not sacrifice much accuracy, as confirmed by our experiments.

The criterion to determine cluster membership is the cumulative squared approximation errors contributed by all the vertices in the row. There are still two alternating steps in the revised CPCA algorithm. In the first step, every row of vertices in an active cluster of segments, $\Omega_{k,g_k^j} (k \in I_b)$, is assigned to the PCA cluster that produces minimal cumulative squared errors. In the second step, the basis vectors of the new PCA clusters are updated. Thus, in this revised algorithm, existing clusters in E_j can accept new members, but their previously existing members cannot change their memberships. The basis vectors of these clusters are updated only when these clusters have accepted a significant number of new members. If we define a cost function as the summed squared approximation errors within all m_{cpf} clusters, it is straightforward to show that this cost function monotonically decreases during each of the above two steps. Therefore, the revised CPCA algorithm converges.

4.2. Reclustering

In the reclustering step, to guarantee high-quality approximation within the per-frame budget, we simply choose to generate m_{cpf} entirely new PCA clusters for the current pose by running the original CPCA algorithm. Before doing that, those rows of vertices associated with the first subgroup of clusters of mesh segments, i.e. $\{\Omega_{k,g_k^j} | k \in I_a\}$, need to be removed from the PCA clusters where they have been previously assigned.

4.3. Cluster Merging

Because reclustering may split among multiple new clusters vertices that previously belong to the same PCA cluster, it may cause the number of PCA clusters used by another pose to exceed our per-frame budget. Therefore, we perform a cluster merging step once our incremental algorithm has generated clusters for all the poses. As a preprocessing step, we first measure distance between pairwise PCA clusters. We only need to consider pairs of clusters that have members from the same pose, and only consider those poses that have used more clusters than they should. The distance beween two PCA clusters depends on the similarity of their corresponding principal component basis, and the similarity of their mean vectors. We measure the distance between two set of principal component basis by the L^2-norm of the difference between their corresponding projection matrices and the difference between their mean vectors [GL96]. The difference between the mean vectors is more elaborated. We only consider the difference between the residue vectors which are the orthogonal part of the mean vector to the subspace spanned by the corresponding PCA basis. If two PCA clusters have similar basis and residue vectors, the subspaces they spanned are similar even though they have different means vectors. The distance between two PCA clusters can thus be formulated as :

$$d(\Psi, \Phi) = \|\mathbf{P}_\Psi - \mathbf{P}_\Phi\|_2 + \eta \|\mathbf{M}_\Psi - \mathbf{M}_\Phi\|_2, \qquad (4)$$

where $\mathbf{P}_\Psi = \mathbf{U}_\Psi \mathbf{U}_\Psi^T$ is the projection matrix of cluster Ψ, $\mathbf{U}_\Psi$ is a matrix whose columns are the basis vectors of Ψ; $\mathbf{M}_\Psi = m_\Psi - \mathbf{P}_\Psi m_\Psi$ is the residue vector, m_Ψ is the mean vector of Ψ, and η is the weighting for residue difference; $\mathbf{P}_\Phi$ and $\mathbf{M}_\Phi$ are defined similarly.

During each iteration of the merging process, a pair of PCA clusters with the minimal distance is chosen. This pair of clusters is merged only if at least one pose shared by their members has more clusters than the per-frame budget. Once merged, distances between the new cluster and all relevant existing clusters need to be computed. This process is repeated until the number of clusters used by all poses has fallen below the upper bound, m_{cpf}. There exist efficient and accurate algorithms for merging two clusters without going back to the raw data in these original clusters. We apply the merging algorithm in [HMM00]. At the end of merging, the coefficient vector of every transfer matrix needs to be recomputed according to its final cluster membership.

5. Runtime Algorithm

The shading equation we follow is largely similar to the one in [SHHS03]. Although our implementation is in three color channels, in the following, we explain the shading process using a single channel. The final radiance from a surface point along a specific viewing direction $\mathbf{V}$ and under a specific low-frequency global lighting vector $\mathbf{L}$ can be formulated as

$$\Gamma = \mathbf{V}^T\mathbf{BTL}, \tag{5}$$

where $\mathbf{B}$ is a BRDF matrix, $\mathbf{T}$ is the radiance transfer matrix with an integrated rotation from the global frame to the local frame [SKS02], and the lighting vector $\mathbf{L}$ is a coefficient vector computed by projecting an environment map onto SH bases. The BRDF matrix is obtained by first discretizing a continuous BRDF, $B(v,s)$, in both viewing and lighting directions to obtain an intermediate matrix whose rows correspond to different viewing directions and columns to different lighting directions. Here we use stereographic projection for our BRDF parameterization. Then each row of the intermediate matrix is projected onto the spherical harmonic (SH) bases.

In this paper, we always use 25 spherical harmonic bases for both lighting and BRDF. Inspired by PCA-based separable approximations of an arbitrary BRDF [KM99], we further factorize the BRDF matrix using singular value decomposition (SVD) and represent it as a product of a view map, $\mathbf{H}$, and a light map, $\mathbf{G}$, $\mathbf{B} = \mathbf{HG}^T$. There is an important distinction between our factorization and that in [LSSS04, WTL04]. We decompose the BRDF matrix after spherical harmonic projection while they directly decompose the original BRDF for all-frequency rendering. Since we have also found experimentally that a 4-term approximation can produce sufficiently accurate results, both $\mathbf{H}$ and $\mathbf{G}$ are chosen to have only four columns. With the resolution of the BRDF view map set to 32×32 and the number of SH bases being 25, $\mathbf{H}$ and $\mathbf{G}$ are represented as 1024×4 and 25×4 matrices, respectively.

Suppose during runtime we need to estimate the transfer matrix $\tilde{\mathbf{T}}_v$ at vertex v for a new incoming pose whose most similar sampled pose is p_j, which is found quickly using a Kd-tree. Let v belongs to the k-th mesh segment, which further belongs to a cluster of mesh segments, Ω_{k,g_k^j}. Then $\tilde{\mathbf{T}}_v$ can be interpolated from the transfer matrices defined at the corresponding vertices of v on the mesh segments in Ω_{k,g_k^j} as follows.

$$\tilde{\mathbf{T}}_v = \sum_l \alpha_l \mathbf{T}_v^{j_l} = \sum_l \alpha_l \left(\sum_i c_i^{v,j_l} \mathbf{U}_i^l \right), \tag{6}$$

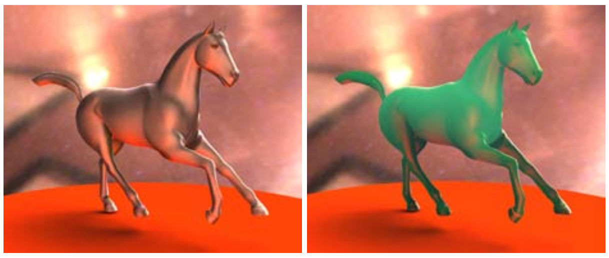


Figure 3: *Renderings from our method. Left: a glossy deforming mesh. Right: a translucent deforming mesh. Both images exhibit global shading effects, including soft shadows, diffuse and specular interreflections. The right image also exhibits subsurface scattering.*

where $\mathbf{T}_v^{j_l}$ represents the transfer matrix at the corresponding vertex of v on a mesh segment in Ω_{k,g_k^j}, and α_l is its interpolation coefficient. The interpolation coefficients are derived from the distance between input pose and the sampled poses in Ω_{k,g_k^j}. This distance computation has non-uniform weights for different bones. We give larger weight to the bone which the mesh segment belong to than other bones. These interpolation coefficients are implemented using normalized radial basis functions (NRBFs) [Nel00], and need to be computed only once for each mesh segment. Since we still use clustered PCA to approximate transfer matrices, $\{\mathbf{U}_i^l | i = 1,\ldots,n_b\}$ represent the set of PCA basis matrices whose linear combination approximates $\mathbf{T}_v^{j_l}$, and $c_i^{v,j_l}(i = 1,\ldots,n_b)$ are the PCA coefficients. Because our revised CPCA algorithm assigns different $\mathbf{T}_v^{j_l}$'s from the same row of Ω_{k,g_k^j} to the same PCA cluster, we can drop the superscript in $\{\mathbf{U}_i^l | i = 1,\ldots,n_b\}$, and (6) becomes

$$\tilde{\mathbf{T}}_v = \sum_i \left(\sum_l \alpha_l c_i^{v,j_l} \right) \mathbf{U}_i = \sum_i \lambda_i \mathbf{U}_i, \tag{7}$$

where $\lambda_i = \sum_l \alpha_l c_i^{v,j_l}$. This means we can simply interpolate scalar PCA coefficients to avoid reconstructing multiple transfer matrices.

Substituting (7) into (5), we obtain

$$\Gamma = \mathbf{V}^T\mathbf{HG}^T \left(\sum_i \lambda_i \mathbf{U}_i \right) \mathbf{L} \tag{8}$$

$$= \left(\mathbf{V}^T\mathbf{H} \right) \overbrace{(\sum_i \lambda_i \underbrace{((\overbrace{\mathbf{G}^T\mathbf{U}_i}^{\mathbf{Q}_i})\mathbf{L})}_{\mathbf{S}_i})}^{\mathbf{Z}}, \tag{9}$$

where the parentheses in (9) indicate the order of evaluation we use, and $\mathbf{Q}_i$, $\mathbf{S}_i$ and $\mathbf{Z}$ are intermediate variables.

The computation of (9) is partitioned into multiple stages, including precomputing, a runtime CPU pass and two runtime GPU passes. We have implemented the GPU passes using DirectX 9 API.

• In the precomputing stage, we multiply the 4×25 matrix, $\mathbf{G}^T$, with each 25×25 matrix, $\mathbf{U}_i$, to obtain a 4×25 matrix, $\mathbf{Q}_i$. Such computation is performed for the 8 basis matrices we use for each PCA cluster. Since the size of $\mathbf{Q}_i$ is smaller than a basis matrix, this step actually helps improve our overall data compression ratios.

• During the runtime CPU pass, we select the $\mathbf{Q}_i$'s needed by the current frame and multiply each of them with the current 25×1 lighting vector, $\mathbf{L}$, to obtain a 4×1 vector, $\mathbf{S}_i$. The time complexity of this step is proportional to the number of PCA clusters needed by the current frame. The resulting vectors, $\mathbf{S}_i$'s, are saved as a 2D texture. In addition, we compute λ_i's by interpolating relevant PCA coefficients as in (7). This step needs to be done for every vertex, and is performed on the CPU because the available bandwidth between the system memory and GPU does not permit us to transmit the data before interpolation. In our experiment, this stage is generally the bottleneck of our rendering algorithm. To improve runtime performance, we have implemented this pass using two threads on a dual-core processor. The resulting coefficients are saved as textures as well. Since only a subset of the clusters are selected in each frame, all the computed textures during this pass become small enough to be transmitted to the GPU on a per-frame basis.

• In the first GPU pass, we compute $\mathbf{Z} = \sum_i \lambda_i \mathbf{S}_i$ for every vertex, where λ_i and $\mathbf{S}_i$ are passed from the previous CPU pass. Although this is purely vertex-based processing, considering the performance limitation of vertex textures on DirectX 9 generation GPUs, we choose to implement it in a pixel shader program via a GPGPU technique by drawing a quad covering as many pixels as the total number of vertices. Every pixel in the quad computes its own $\mathbf{Z}$ vector using the textures representing $\mathbf{S}_i$'s and λ_i's. These $\mathbf{Z}$ vectors from the pixel shader are saved back to the GPU video memory as a 2D texture map using multiple render targets.

• In the second GPU pass, we use a pixel shader program to first compute $\mathbf{V}^T\mathbf{H}$ for every pixel. Since we use bilinear interpolation in the viewing direction, $\mathbf{V}$ has four nonzero entries, and $\mathbf{V}^T\mathbf{H}$ is computed as a linear blend of four row vectors of $\mathbf{H}$. Meanwhile, the per-vertex $\mathbf{Z}$ vectors from the previous pass are linearly interpolated at every pixel in the GPU pipeline implicitly. At the end, we perform pixelwise multiplication between the row vector, $\mathbf{V}^T\mathbf{H}$, and the interpolated $\mathbf{Z}$ vector to obtain the final radiance for each color channel.

In addition to opaque surfaces, we also support translucent objects with subsurface scattering (Fig. 3). We follow the algorithms in [WTL05] for precomputing single and multiple scattering. The precomputed results can still be represented using transfer matrices except that the BRDF matrix in (5) should be replaced with a matrix that accounts for light coming from both sides of the surface.

	Armadillo	Boxer	Horse
original data size	253GB	215GB	3.06GB
compressed data size	1.7GB	1.54GB	76MB
data used in demo	200MB	430MB	76MB
frame rate	30	30	45∼50

Table 2: *Compression Result and Performance. As shown, we have achieved a compression ratio of around 140 on large examples. Because we generated the poses with enough variations, the demo animations from our paper usually require only a small subset of the sampled poses for real-time interpolation. The Cook-Torrance BRDF model is used for Armadillo and the Phong model is used for both Boxer and Horse. All performance measurements were taken from a 3.0GHz PentiumD^{TM}with nVidia Geforce 7900GTX^{TM}512MB VRAM.*

6. Experimental Results

We have successfully experimented with three examples. Renderings from our method can be found in Figs. 1, 3 and 4. For each example, we start with a static mesh and a few deformed versions of this mesh. The deformed meshes for Boxer and Armadillo model are obtained from the results in [SYBF06]. A skinning model with blending weights is trained from these deformed versions using the technique in [JT05], which also produces a consistent segmentation of the meshes. Meanwhile, we obtain a number of MoCAP sequences such as walking, running, boxing and dancing, from the CMU database [MoC], and align the skeleton used in the MoCAP sequences with the segmented meshes. As discussed in Section 2, CPCA clustering is then performed on these pose data to generate the pose subspace clustering. We then resample new poses by adjusting the PCA coefficients in each pose subspace cluster to obtain a database of sampled poses. This is a fairly automatic process. We do not perform any other pose selection except we only generate up to 1024 poses, which are evenly distributed among the pose clusters. Each of the sampled poses can generate a deformed version of the original mesh using the trained skinning model, and we precompute radiance transfer matrices for each of them using ray-tracing [PH04]. The statistics of these precomputed datasets are shown in Table 1 and the compression results are in Table 2. As we can see, two of the examples are very large. Each of them has more than 200GB of raw PRT data. Our data processing algorithms actually divide such large-scale raw data into smaller chunks and only load into the memory one chunk at a time. Once we have run the mesh segment clustering algorithm and the revised CPCA algorithm, we quantize each PCA coefficient down to 16 bits. In this way, we can achieve a compression ratio of around 140 without losing much visual fidelity. Although the number of generated poses are relatively large, only a fraction of the sampled poses might be used during the runtime interpolation. As shown in Table 2, the demo videos in

	#vertices	#poses	# mesh segs	prt coefficients computation time	segment clustering		revised CPCA	
					#clusters	cpu time	#clusters	cpu time
Armadillo	33,000	1024	19	77hrs	2432	42hrs	36000	91hrs
Boxer	32,000	1024	11	64hrs	2816	38hrs	96000	115hrs
Horse	19,000	48	27	2hrs	648	30mins	2984	1.7hrs

Table 1: *Statistics for Precomputation. We utilize the cluster servers in our institution to accelerate the computation for PRT coefficients. In the Boxer example, every mesh segment cluster contains 4 segments. There are 72000 PCA clusters on the character model and additional 24000 clusters for a floor plane. In the Armadillo example, every mesh segment cluster contains 8 segments, which have the side effect that the total number of PCA clusters is much reduced. The small number of sampled poses for the Horse example gives rise to much smaller numbers of mesh segment clusters and PCA clusters, and an overall much smaller dataset. Every PCA cluster in these examples has eight* 25×25 *basis matrices. The number of vertices in Boxer and Horse examples also include the ground plane.*

	Our Scheme	Pose-Space
1	8851	10661
2	5307	11015
3	6104	8167
4	8348	9686
5	9819	13922
Avg	7685.2	10690.2

Table 3: *Comparisons of approximation errors between our interpolation scheme for transfer matrices and pose-space based interpolation. The first five rows show the errors for five randomly generated poses, and the last row shows the average errors among the five.*

our paper make use only a small subset of the sample poses and require less than 512MB of memory.

With such a precomputed and compressed dataset, we can render dynamically deformed versions of the original mesh in real time. Our test system is a 3.0GHz PentiumD$^{\mathrm{TM}}$[†] computer with nVidia Geforce 7900GTX$^{\mathrm{TM}}$[‡]512MB VRAM. We have experimented with two possible methods to generate dynamically deformed meshes. In the first method, the user can interactively adjust the pose of the skeleton, and every adjusted pose produces a new deformed mesh. In the second method, we can take a new MoCAP sequence with the same skeletal structure and use the poses in this sequence to deform the mesh. Note that these poses are different from all previously sampled poses. In both cases, the transfer matrices for the dynamically deformed mesh are interpolated from the precomputed data in real time and the deformed mesh is shaded instantly on the GPU using the interpolated transfer matrices. Since it is not very convenient to interactively produce interesting new poses without referencing MoCAP data, most of our demonstrations use the second method.

[†] PentiumD is the trademark of Intel Corporation

[‡] Geforce 7900GTX is the trademark of nVidia Corporation

6.1. Validation

6.1.1. Frame Rate

Frame rate is by far the most important goal. An important reason that we can achieve real-time performance with at least 30 frames per second is that transfer matrix interpolation is actually performed through the interpolation of their scalar PCA coefficients, as formulated in (7). This is facilitated by both mesh segment clustering and the revised CPCA algorithm. We did a comparison on the two large examples between our technique and a different version that does not use the same set of PCA bases to approximate transfer matrices at corresponding vertices within the same cluster of mesh segments. The latter only achieved at most 8 frames per second, more than three times slower than our version.

6.1.2. Interpolation Accuracy

We interpolate transfer matrices for each mesh segment independently using the cluster of mesh segments it belongs. Our clusters of mesh segments are formed considering similarity in both transfer matrices and pose. In contrast, given a new pose, a conventional scheme finds a few nearest sampled poses, and then simply interpolates all transfer matrices for the new pose from the same set of nearest sampled poses. It is thus a global scheme that always uses the same set of neighbors for different mesh segments.

We have compared our interpolation scheme with this pose-space scheme on our examples. During the precomputing stage of these comparisons, we use the same number of PCA clusters to compress the raw PRT data in both cases. During runtime, we randomly generate new poses that further produce new deformed meshes, which are rendered using the two interpolation schemes which choose different sets of neighbors. We also directly ray-trace the deformed meshes to produce the ground truth. Table 3 lists summed errors of the interpolated transfer matrices for a few randomly generated poses. The errors of our interpolation scheme are consistently lower. However, differences in numerical errors may not reflect the true differences in visual quality. We further compare the rendered images from these interpolation

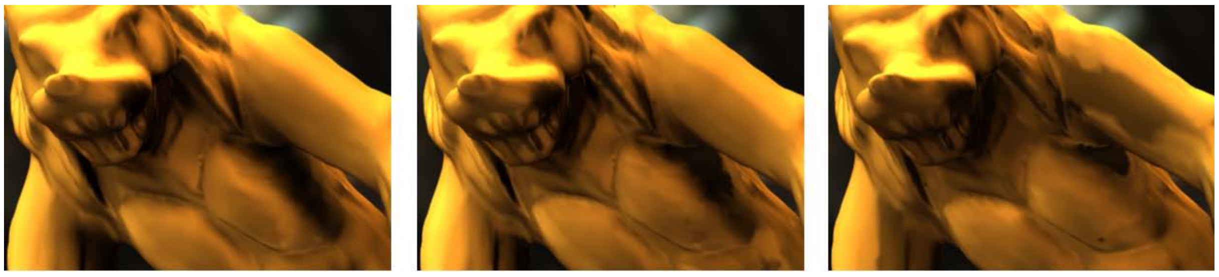


Figure 5: *Another comparison between our interpolation scheme and pose-space based transfer matrix interpolation in the self-occlusion area. Pose-space interpolation fails to capture some self-occluded effects and has more visual artifacts*

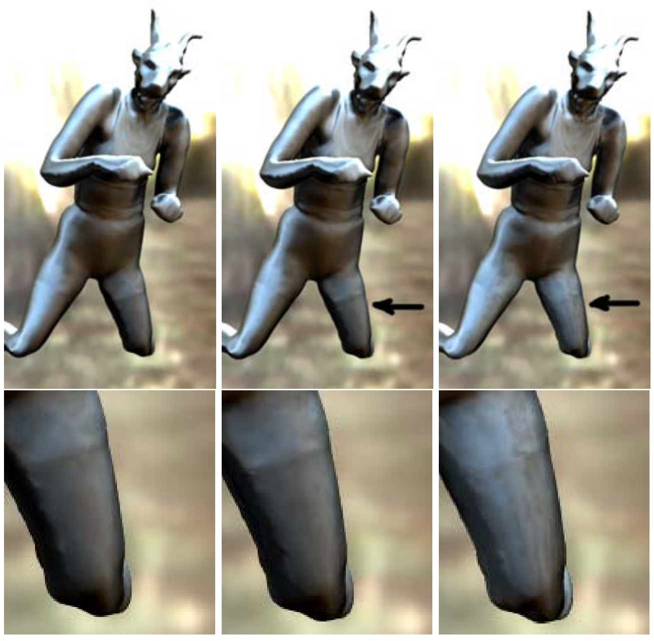


Figure 4: *A comparison between our interpolation scheme and pose-space based transfer matrix interpolation. Since pose-space interpolation only considers similarity in global pose configurations without accounting for similarity among transfer matrices themselves, visually noticeable artifacts occur.*

schemes in Figs. 4 and 5. The global scheme produces obvious artifacts on certain parts of the surface while the results from our scheme are visually comparable to the ground truth.

6.1.3. Compression Quality

We further validate the compression quality of our revised CPCA algorithm by comparing it with running the original CPCA on each pose independently. Because an important goal of our algorithm is to satisfy the two requirements discussed in Section 4 instead of achieving maximum compression ratios, without incremental cluster creation, it should generate larger approximation errors than the original CPCA. However, incremental cluster creation exploits data redundancies across different poses and, therefore, can compensate the negative effects produced by the requirements we impose on compression. As a result, with the same total number of PCA clusters, our revised CPCA actually achieves smaller approximation errors than running CPCA independently on each pose. For example, on the BOXER dataset, when there are a total of 72000 clusters over 1024 poses, the average error per pose is 4489 for our algorithm and 5516 for independent CPCA. More importantly, the latter cannot satisfy the second requirement, therefore, would significantly lower the frame rate. As shown in Fig. 6, our revised algorithm also produces results with better visual quality and without obvious boundaries among PCA clusters.

7. Conclusions and Future Work

We have presented effective data clustering and compression techniques as well as an efficient runtime algorithm that can achieve high-quality real-time rendering of dynamically skinned models using precomputed radiance transfer. Our techniques can reduce the amount of precomputed data to a managable size, and achieve a compression ratio of 140 on large-scale datasets with hundreds of gigabytes of raw data. Meanwhile, they also facilitate runtime data communication, decompression and interpolation. Our algorithms and results have demonstrated that using an example-based approach for PRT-based rendering of dynamic objects with glossy or translucent materials is both feasible and practical.

There are limitations with our current algorithms and implementation. First, we are limited to low-frequency environment lighting. We would like to investigate in future whether it is feasible to extend our work to all-frequency lighting using nonlinear wavelet approximation [NRH03, NRH04]. Second, we would like to model interactions among multiple objects using subspaces and investigate precomputed radiance transfer for such interactions. Third, our implementation is partially limited by the memory capacity and streaming bandwidth of the current generation of GPUs.

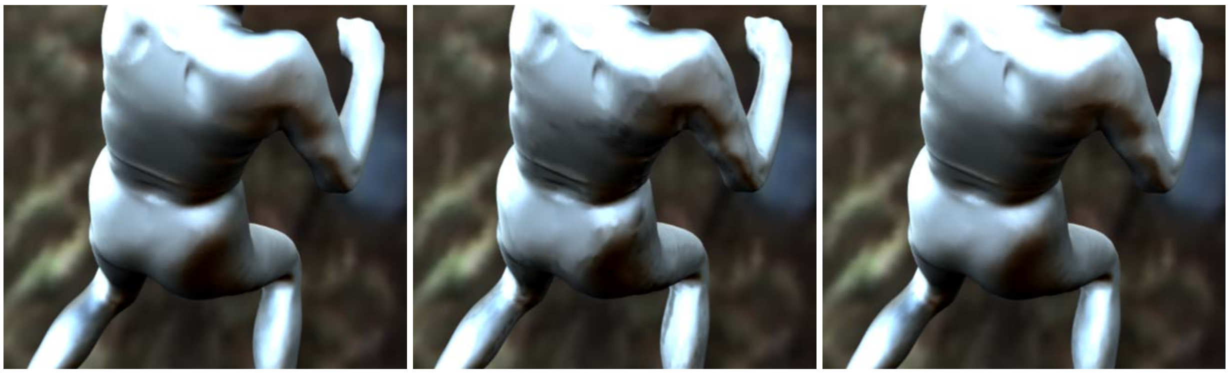


Figure 6: *A comparison between our revised CPCA and running CPCA independently on different poses. Note that our revised CPCA satisfies additional requirements, and incrementally creates new clusters. With the same total number of PCA clusters (72000 clusters over 1024 poses in this example), our algorithm produces visually better results while independent CPCA produces visible boundary effects.*

We expect these aspects improved in the future generations and even better runtime performance achieved on them.

Acknowledgments

We wish to thank Rui Wang for sharing his code for precomputing subsurface scattering, Lin Shi for sharing his mesh animation datasets, and the anonymous reviewers for their valuable comments. This work was partially supported by a gift from Rambus, Inc. Yuntao Jia was supported by the R.T.Cheng Fellowship.

References

[Ari06] ARIKAN O.: Compression of motion capture databases. *ACM TOG 25*, 3 (2006), 890–897.

[Bra02] BRAND M.: Incremental singular value decomposition of uncertain data with missing values. In *Proc. European Conference on Computer Vision (Vol. I)* (2002), pp. 707–720.

[GL96] GOLUB G., LOAN C. V.: *Matrix Computations*. The Johns Hopkins Unversity Press, Baltimore, third edition, 1996.

[HMM00] HALL P., MARSHALL D., MARTIN R.: Merging and splitting eigenspace models. *IEEE Trans. Pattern Analysis and Machine Intelligence 22*, 9 (2000), 1042–1049.

[HPB06] HASAN M., PELLACINI F., BALA K.: Direct-to-indirect transfer for cinematic relighting. *ACM TOG 25*, 3 (2006), 1089–1097.

[JF03] JAMES D., FATAHALIAN K.: Precomputing interactive dynamic deformable scenes. *ACM TOG 22*, 3 (2003), 879–887.

[JT05] JAMES D., TWIGG C.: Skinning mesh animations. *ACM TOG 24*, 3 (2005), 399–407.

[KA06a] KIRK A., ARIKAN O.: Precomputed ambient occlusion for character skins. SIGGRAPH 2006 Sketches, 2006.

[KA06b] KONTKANEN J., AILA T.: Ambient occlusion for animated characters. In *Rendering Techniques 2006 (Eurographics Symposium on Rendering)* (jun 2006), Wolfgang Heidrich T. A.-M., (Ed.), Eurographics.

[KA07] KIRK A. G., ARIKAN O.: Real-time ambient occlusion for dynamic character skins. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (April 2007).

[KAMJ05] KRISTENSEN A., AKENINE-MÖLLER T., JENSEN H.: Precomputed local radiance transfer for real-time lighting design. *ACM TOG 24*, 3 (2005), 1208–1215.

[KM99] KAUTZ J., MCCOOL M.: Interactive rendering with arbitrary brdfs using separable approximations. In *Eurographics Workshop on Rendering* (1999), pp. 281–292.

[LCF00] LEWIS J., CORDNER M., FONG N.: Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Computer Graphics Proceedings, Annual Conference Series* (2000).

[LSSS04] LIU X., SLOAN P.-P., SHUM H.-Y., SNYDER J.: All-frequency precomputed radiance transfer for glossy objects. In *Eurographics Symposium on Rendering* (2004), pp. 337–344.

[MBLS01] MALIK J., BELONGIE S., LEUNG T., SHI J.: Contour and texture analysis for image segmentation. *Int'l Journal of Computer Vision 43*, 1 (2001), 7–27.

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM TOG 22*, 3 (2003), 562–568.

[MoC] MOCAP L.: CMU graphics lab motion capture database. http://mocap.cs.cmu.edu.

[Nel00] NELLES O.: *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer Verlag, 2000.

[NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM TOG 22*, 3 (2003), 376–381.

[NRH04] NG R., RAMAMOORTHI R., HANRAHAN P.: Triple product integrals for all-frequency relighting. *ACM TOG 23*, 3 (2004), 477–487.

[PH04] PHARR M., HUMPHREYS G.: *Physically Based Rendering*. Morgan Kaufmann, 2004.

[RWS*06] REN Z., WANG R., SNYDER J., ZHOU K., LIU X., SUN B., BAO P.-P. S. H., PENG Q., GUO B.: Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM TOG 25*, 3 (2006), 977–986.

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM TOG 22*, 3 (2003), 382–391.

[SHP04] SAFONOVA A., HODGINS J., POLLARD N.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM TOG 23*, 3 (2004), 512–519.

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precompted radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM TOG 21*, 3 (2002), 527–536.

[SLS05] SLOAN P.-P., LUNA B., SNYDER J.: Local, deformable precomputed radiance transfer. *ACM TOG 24*, 3 (2005), 1216–1224.

[SM00] SHI J., MALIK J.: Normalized cuts and image segmentation. *IEEE Trans. Pat. Anal. Mach. Intell. 22*, 8 (2000), 888–905.

[SM06] SUN W., MUKHERJEE A.: Generalized wavelet product integral for rendering dynamic glossy objects. *ACM TOG 25*, 3 (2006), 955–966.

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast multigrid algorithm for mesh deformation. *ACM Transactions on Graphics 25*, 3 (2006), 1108–1117.

[TS06] TSAI Y.-T., SHIH Z.-C.: All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation. *ACM TOG 25*, 3 (2006), 967–976.

[WTL04] WANG R., TRAN J., LUEBKE D.: All-frequency relighting of non-diffuse objects using separable brdf approximation. In *Eurographics Symposium on Rendering* (2004), pp. 345–354.

[WTL05] WANG R., TRAN J., LUEBKE D.: All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM TOG 24*, 3 (2005), 1202–1207.

[ZHL*05] ZHOU K., HU Y., LIN S., GUO B., SHUM H.: Precomputed shadow fields for dynamic scenes. *ACM TOG 24*, 3 (2005), 1196–1201.