Real-Time Ambient Occlusion for Dynamic Character Skins

Adam G. Kirk*
UC-Berkeley

Okan Arikan†
UT-Austin

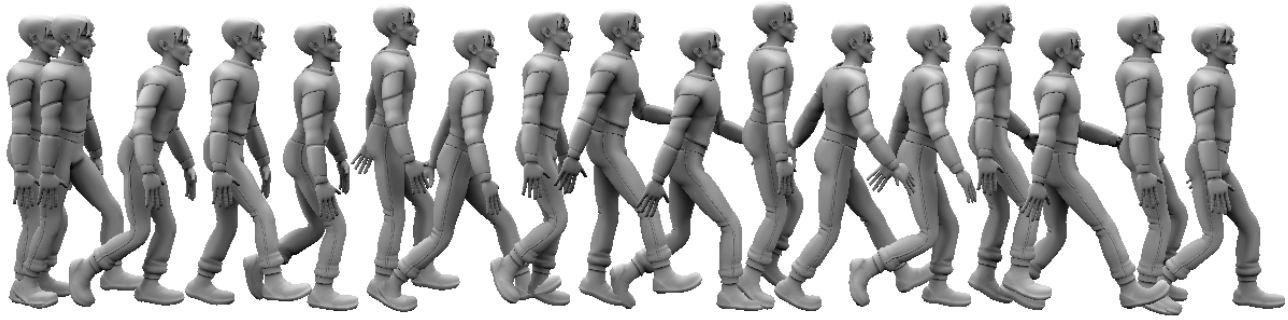

Figure 1: Time-lapse screen capture showing ambient occlusion values calculated in real-time.

Abstract

We present a single-pass hardware accelerated method to reconstruct compressed ambient occlusion values in real-time on dynamic character skins. This method is designed to work with meshes that are deforming based on a low-dimensional set of parameters, as in character animation. The inputs to our method are rendered ambient occlusion values at the vertices of a mesh deformed into various poses, along with the corresponding degrees of freedom of those poses. The algorithm uses k-means clustering to group the degrees of freedom into a small number of pose clusters. Because the pose variation in a cluster is small, our method can define a low-dimensional pose representation using principal component analysis. Within each cluster, we approximate ambient occlusion as a linear function in the reduced-dimensional representation. When drawing the character, our method uses moving least squares to blend the reconstructed ambient occlusion values from a small number of pose clusters. This technique offers significant memory savings over storing uncompressed values, and can generate plausible ambient occlusion values for poses not seen in training. Because we are using linear functions our output is smooth, fast to evaluate, and easy to implement in a vertex or fragment shader.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1 Introduction

Character animation is ubiquitous in computer graphics. The most obvious examples are found in video games and motion pictures. Realistic character shading, or global illumination, improves the three-dimensional quality of the character and helps the viewer place the character in the scene. Unfortunately, computing global illumination is a time-consuming process.

Ambient occlusion has emerged as a viable alternative to a full global illumination solution[Landis 2002][Christensen 2002]. Ambient occlusion at a point is the cosine weighted integral of the visibility function. Assuming the scene is enclosed in a spherical area source light, ambient occlusion corresponds to the amount of light that will reach each point. While computing ambient occlusion is less expensive than computing global illumination, it is still too expensive to be solved for in real-time. This is due to the fact that it requires sampling the hemisphere around each point to determine visibility.

Recent advances in data-driven character animation have shown pose subspaces to be a useful tool in manipulating and compressing motion information[Chai and Hodgins 2005][Arikan 2006]. Pose subspaces represent motion data as points in a set of linear subspaces. Poses that are close together are represented as points in the same linear subspace, while poses that are significantly different are defined as points in separate subspaces. Our paper examines the benefits of using pose subspaces to specify ambient occlusion.

Our method takes a data-driven approach to approximating ambient occlusion in real-time. It builds a set of linear functions over several subsets of poses, then produces ambient occlusion values using a blend of these functions. Because we are using linear functions, our output is smooth, fast to evaluate, and easy to implement in a vertex or fragment shader.

2 Related Work

Recently, several groups have examined hardware accelerated ambient occlusion calculation. [Sattler et al. 2004] renders data from multiple directional light sources into depth maps to calculate ambient occlusion. Similarly, [Bunnell 2005] describes a method for calculating ambient occlusion and indirect lighting in real-time. This algorithm works in multiple passes of hardware. Each pass is designed to reduce the error in the approximation. In contrast, our method pushes much of the work to the precomputation stage, so evaluation consists of a fast, single pass.

Precomputing ambient occlusion values allows for more efficient rendering. [Kontkanen and Laine 2005] precomputes ambient oc-

*email:akirk@cs.berkeley.edu
†email:okan@cs.utexas.edu

clusion fields for static objects as a function of shape and distance from the object. Their method is similar to ours in that it fits a model using least-squares. However, they use a spherical cap approximation to the hemispherical shape projection and are unable to handle deforming objects. [Hegeman et al. 2006] presents an approximate method for calculating ambient occlusion for trees. Their method requires no precomputation and runs in real-time, but it relies on certain properties unique to foliage. Our method is most similar to the algorithm described in [Kontkanen and Aila 2006]. In this work, the authors also express ambient occlusion as a function of pose. The major difference between the two methods is that their method builds a function over the entire space of character poses. In contrast, our method learns a multilinear model, where each model is localized to a different region of pose space. This allows the user a tradeoff between compression and accuracy, and also scales better to large datasets. Furthermore, the pose representation used in our method is a reduced-dimensional set of handle positions, while poses in their method are the set of joint angles. As well as being smaller, our pose representation exists in a uniform space. Finally, we describe a method for efficiently compressing our representation of ambient occlusion spatially. This technique clusters vertices on the mesh into groups with similar ambient occlusion values over various regions of pose space, then represents ambient occlusion for all vertices in a group with a single coefficient vector. The major advantage of our method is in rendering time. By compressing our pose representation, we are able to significantly decrease the amount of work required per vertex, at the cost of a handful of matrix multiplies for each frame rendered.

Several researchers have examined generating real-time illumination values for dynamic lighting. [Sloan et al. 2002] introduces precomputed radiance transfer (PRT). This method compresses low frequency transfer functions using spherical harmonics, and handles a wide range of low-frequency phenomena. [Ng et al. 2004] presents triple product wavelet integrals to model light transport functions. These wavelet integrals handle all frequencies, and allow the user to change light and viewing directions. However, these methods only apply to static scenes. Recently, [Sloan et al. 2005] extends PRT to work with locally deforming geometry. Using this technique, changes to the mesh in the local region of the vertex being shaded will change the shading, but distant changes to the mesh are unable to influence shading. While our method does not compress a function as complicated as the transfer function, it is designed to handle changes in ambient occlusion due to large-scale changes in the mesh. [Ren et al. 2006] describes a method for computing soft shadows in real-time for dynamic scenes. Their method approximates geometry with sets of spheres, then represents the low-frequency visibility functions at various points using spherical harmonics. These visibility functions are then combined using spherical harmonic exponentiation.

Our work has similarities to the techniques used in [James and Fatahalian 2003] and [James and Twigg 2005]. These methods store illumination data at various points in state space, then compress this data using principal component analysis (PCA). To evaluate shading at a point in state space, they interpolate between nearby illumination points. Rather than using PCA to compress illumination data, our method uses PCA to find a lower-dimensional subspace in which to represent poses. Within each of the low-dimensional pose subspaces, our method then builds a function that expresses ambient occlusion in terms of pose. This provides a means to minimize variance within local regions of state space. By minimizing variance within a pose subspace, we are able to accurately represent ambient occlusion using fewer subspaces. Of course, we have the additional problem of moving between pose subspaces, but moving least squares [Levin 1998][Shen et al. 2004] smoothes any discontinuities.

Our method builds local subspaces of pose data to efficiently compress ambient occlusion data. Several other researchers have used local subspaces of pose data to accomplish different goals. In particular, [Chai and Hodgins 2005] learn a low-dimensional space for control signals derived from video. The subspace is determined at query time. Due to the amount of raw data, we define local subspaces during preprocessing. Similarly, [Arikan 2006] defines motion subspaces in order to compress the motion data itself. Velocity information is used in both of these methods to define local subspaces. While our method has a similar notion of subspaces, they are defined using only position data. This is because illumination is strictly a function of position.

Figure 2: The ambient occlusion computed with our method is applied to a colored material.

3 Methods

Our method computes ambient occlusion values in real-time on dynamic character skins. Character skins have the property that they are controlled by a low number of parameters, namely joint angles (one popular method for controlling character skins is linear blend skinning). This parameterization distinguishes character skins from general deforming meshes. While the skin changes significantly between poses, it does so in a well-defined parameter space. Furthermore, ambient occlusion on the mesh is a function of triangle position. Triangle position, in turn, is a function of joint angles. Therefore, it is reasonable to express ambient occlusion at a point as a function of joint angles. Joint angles, however, do not provide a uniform space in which to compare poses. For example, a small change in the character's root orientation can produce a large change in the position of the character's hand. Instead of joint angles, we represent poses with handle positions. We place a handle at the joint positions, plus a single handle per bone that is offset from the bone axis, see Figure 4. This defines a coordinate frame for each bone, and fully specifies the position and orientation of the

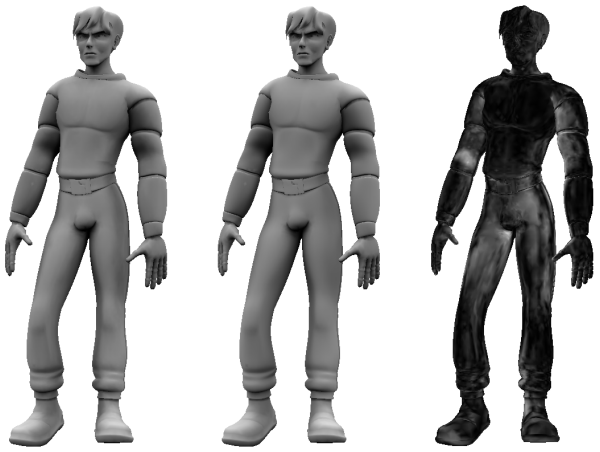

Figure 3: Comparison of ground truth and results produced with our method. On the left is the ground truth data. The uncompressed ambient occlusion values for the entire dataset of 4130 frames is approximately 592MB. In the middle is the result of our method. This image was produced using 10 clusters and 15 dimensions per cluster, with no spatial clustering. Our representation is approximately 23MB. On the right is the magnified error. Each vertex is shaded by the scaled absolute value of the difference between ground truth and our method. The values are scaled by 10.

bone in space. The handle representation is a more uniform space than joint angles for representing pose [Arikan 2006]. In this paper we define pose to be the vector of handle positions at a frame of motion data.

We take a data-driven approach to compressing ambient occlusion values for character skins. Given a set of ambient occlusion values and the corresponding poses, our method builds a set of linear functions that express ambient occlusion at a point in terms of pose. Our training data consists of ambient occlusion values at the vertices of the skin, for a set of poses taken from a motion capture database.

3.1 Function Approximation

Our method specifies ambient occlusion at a point on the mesh as a function of pose. Ambient occlusion can change nonlinearly over a wide range of poses. However, a key observation is that ambient occlusion at a point on the mesh changes smoothly for small changes in pose. Therefore, if we decompose ambient occlusion over the set of poses, we can model this function linearly. Our method builds several linear functions, each one local to a different region of pose space. This is beneficial because linear functions change smoothly, are fast to compute, and are easy to implement in a hardware shader.

The domain of each locally linear approximation is a set of poses that are close together. Because ambient occlusion changes smoothly as pose changes, it follows that poses close together should have similar ambient occlusion values. This observation is the basis of our method. In fact, our method assumes that ambient occlusion and pose are linearly related, at least within local regions. We use clustering techniques to find these local regions. Because pose and ambient occlusion are strongly related, we can cluster on either of these features. The vector of ambient occlusion values at a particular frame is very large, however the vector of handle positions is relatively small. Therefore, for computational efficiency, we only consider pose when defining the domain of each locally linear approximation. Our method uses k-means clustering on the poses in our training set to define these clusters in pose space.

Within a pose cluster, all the poses are relatively close together. Furthermore, representing pose using handle positions (which define a coordinate frame at each bone) encodes redundant information. Our method uses PCA to reduce the dimensionality of the pose vector, which provides significant memory savings by taking advantage of this proximity and redundancy. More specifically, for each pose cluster we compute and store the mean pose. We then compute the n largest principal component vectors, where n is a parameter to our system. We project each of our poses into the subspace represented by these n principal component vectors. This process results in a pose vector with much lower dimensionality. Our method specifies ambient occlusion at a point as a linear function of these reduced dimension pose vectors. For information on how clustering and dimension reduction affect the resulting function, see Table 1.

In addition to building different functions for each pose cluster, our method also builds different functions for separate spatial regions. This is because there is a nonlinear relationship between the ambient occlusion values at different points on the mesh for the same pose. Therefore, our method builds a linear function relating reduced dimension pose to ambient occlusion for each skin vertex in each pose cluster. If there are m pose clusters and n vertices, this results in mn functions. While this may seem prohibitive, note that each of these functions has a small number of coefficients (equal to the size of the reduced dimension pose). Section 3.2 will discuss additional compression of these vectors.

Fitting the linear function is done by minimizing the least-squares error, using a pseudo-inverse. Let $\bar{\phi}_i$ be the mean ambient occlusion value for vertex i across all frames in the training set. Let $1 \ldots k \ldots K$ be the index of the K poses in pose cluster j. Note that there is no notion of temporal ordering, the poses can be ordered in any arbitrary way. Let h_k and $\bar{h}_j$ be the kth pose and mean pose of cluster j, respectively. Let $\phi_{i,k}$ be the ambient occlusion value at vertex i for pose k, and P_j be the matrix that projects the pose vector onto the reduced principal components. The linear coefficient vector $x_{i,j}$ is computed using the pseudo-inverse of the matrix of projected poses, as such:

$$\begin{bmatrix} \phi_{i,1} - \bar{\phi}_i \\ \vdots \\ \phi_{i,K} - \bar{\phi}_i \end{bmatrix} = \begin{bmatrix} (P_j(h_1 - \bar{h}_j))^T \\ \vdots \\ (P_j(h_K - \bar{h}_j))^T \end{bmatrix} x_{i,j} \quad (1)$$

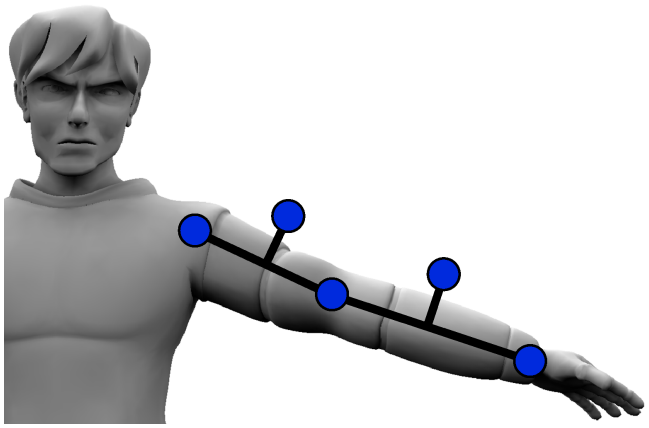

Figure 4: Pose handles for the left arm are shown as blue circles.

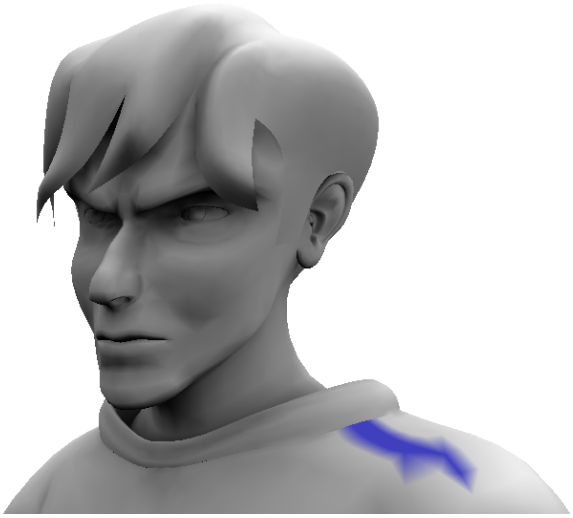

Figure 5: The blue patch on the shoulder shows a sample vertex cluster. This is one of 5000 clusters. The vertices in this cluster have similar ambient occlusion values over the pose cluster from which this frame was taken. Rather than storing a coefficient vector for each vertex, our method stores a single coefficient vector for the vertex cluster.

3.2 Spatial Compression

Our method can optionally perform spatial compression on the coefficient vectors $x_{i,j}$. In the limit, as described above, a coefficient vector is stored for every vertex in every pose cluster. However, there is significant redundancy in these coefficient vectors. The ambient occlusion at vertices in a local neighborhood tends to change similarly with changes in pose. Therefore, we can group these vertices together and store a single coefficient vector for the group. Note that our method computes pose clusters first, then the optional vertex clusters. This allows a different spatial compression within each pose cluster.

We use a clustering procedure to find groups of vertices that have similar mean-subtracted ambient occlusion values for a given pose cluster. The feature vector for a vertex is composed of its mean subtracted ambient occlusion values for each pose in the pose cluster. We expect there to be many clusters of vertices (see Table 1 for details). This is because we are searching for groups of vertices that have similar ambient occlusion values across the entire pose cluster. We found that, due to the large number of clusters, using k-means to cluster vertices is unstable and computationally expensive. Instead, our method uses a greedy, hierarchical clustering technique. To begin, all vertices are placed in the same vertex cluster. The vertex cluster center is computed, and the vertex that is farthest from the center becomes its own cluster. Then the vertices are reassigned to the nearer of the two clusters based on distance to the cluster center. This process is recursively applied to each of the new vertex clusters until the greatest distance between a vertex and its cluster center falls below a threshold, or the maximum number of vertex clusters are created.

At this point, rather than finding a coefficient vector for each vertex as in Section 3.1, our method computes a single coefficient vector for each vertex cluster. This is done by solving for the best fit vector for all vertices in a spatial cluster simultaneously. Let $v_1 \dots v_l$ be the mean subtracted ambient occlusion vectors for all l vertices in spatial cluster i and pose cluster j. Each of these vectors corresponds to the vector in the left hand side of Equation 1. Let Q_j be the reduced-dimensional mean subtracted pose vectors for all poses in pose cluster j. This matrix corresponds to the matrix on the right hand side of Equation 1. To solve for the coefficient vector $x_{i,j}$ for spatial cluster i and pose cluster j, solve

$$\begin{bmatrix} v_1 \\ \vdots \\ v_l \end{bmatrix} = \begin{bmatrix} Q_j \\ \vdots \\ Q_j \end{bmatrix} x_{i,j} \tag{2}$$

For each vertex in each pose cluster, it is necessary to store identification information regarding in which vertex cluster that vertex is a member. Because this is a greedy algorithm, it is non-optimal. It is possible another clustering algorithm could obtain better results. The main benefits of this method include speed and its deterministic nature.

3.3 Function Evaluation

Section 3.1 describes a method to specify ambient occlusion as a set of linear functions described over pose subspaces. Evaluating at a specific vertex for a specific pose is a linear operation. To determine which subspace the query pose h belongs to, simply find the pose cluster center to which h is closest. Let subspace j be the nearest subspace to h. Within subspace j, the reconstructed ambient occlusion value $\tilde{\phi}_i$ at vertex i and pose h is

$$\tilde{\phi}_i = \bar{\phi}_i + (h - \bar{h}_j)^T P_j^T x_{i,j} \tag{3}$$

where $\bar{\phi}_i$ is the mean ambient occlusion value across all poses in the training data for vertex i, $\bar{h}_j$ is the cluster center for cluster j, P_j is the matrix that projects the pose into the reduced dimension pose space j, and $x_{i,j}$ is the coefficient vector computed in Equation 1. If spatial compression (Section 3.2) is used, then $x_{i,j}$ refers to the coefficient vector for vertex cluster i and pose cluster j. When calculating the ambient occlusion at a specific vertex, it is necessary to lookup the coefficient vector for the vertex cluster in which the specific vertex is a member.

However, when computing ambient occlusion for a motion sequence, the character will transition between several subspaces through the course of the motion. Our method uses moving least squares [Levin 1998] [Shen et al. 2004] to eliminate discontinuities in ambient occlusion values as the character moves between these subspaces. Using this technique, ambient occlusion is the weighted sum of the ambient occlusion values computed in each pose subspace:

$$\tilde{\phi}_i = \bar{\phi}_i + \sum_j \omega(h,j)(h - \bar{h}_j)^T P_j^T x_{i,j} \tag{4}$$

where the blend function ω is defined to be

$$\omega(h,j) = \frac{\frac{1}{\|h-\bar{h}_j\|_2^\alpha}}{\sum_k \frac{1}{\|h-\bar{h}_k\|_2^\alpha}} \tag{5}$$

α is a dataset dependent scalar that controls the falloff of the blend function.

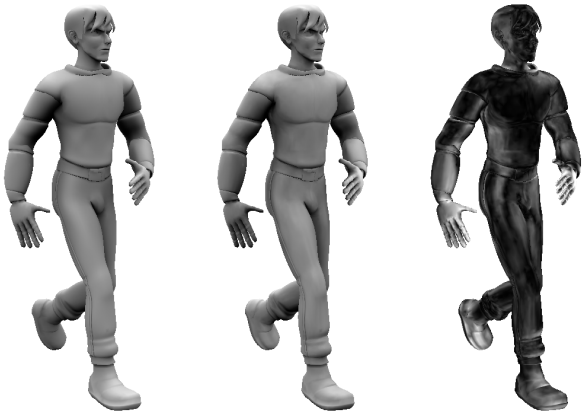

Figure 6: Example of ambient occlusion function generalization. On the left is ground truth for a running motion. In the middle is the output of our method. Our method was trained on walking and skipping motions, but not running. On the right is the magnified error. Each vertex is shaded by the scaled absolute value of the difference between ground truth and our method. The values are scaled by 5.

Regarding efficiency, note that the matrix multiplication in Equation 4 is done once per cluster. While we need to evaluate this equation for each vertex i, we can precompute and store $(h-\bar{h}_j)^T P_j^T$ for each pose cluster j with a non-zero blend function, then reuse it for all vertices. This leaves only a low-dimensional dot product to the per vertex operations. This dot product can easily be implemented in a hardware shader.

4 Results

Our dataset consists of 4130 frames of motion capture data, containing walking, running, skipping, reaching, and crouching motions. The most computationally expensive part of our method was rendering ambient occlusion values. To render, our method sampled the hemisphere at each vertex in each frame using 500 rays. The ambient occlusion at these poses was computed and stored over the course of several days. Once the ambient occlusion values had been calculated, we experimented with several values of our parameters. Each experiment took approximately 15 to 20 minutes. The uncompressed ambient occlusion values for the entire set is approximately 592 megabytes in size. This is approximately 0.14MB per frame, which corresponds to storing one floating point number for each vertex. Our method easily compresses this to less than 40 megabytes. Table 1 summarizes effects due to changes in the number of spatial clusters per pose cluster, for a set number of pose clusters and reduced dimensions. Ambient occlusion is defined to be a number between zero and one. In general, note that the mean error is quite low even with a function representation less than five megabytes in size. Furthermore, the error decreases as the number of pose clusters, number of dimensions, and number of spatial clusters increase. For further evaluation, please see the accompanying video, which was rendered using an ATI Radeon 800GT.

5 Discussion

Our method has four parameters: the number of clusters, the dimensionality of each cluster, the number of spatial cluster vectors per pose cluster, and the moving least squares blend parameter. For information on the first three, see Table 1. The moving least squares blend parameter (α in Equation 5) should be chosen such that a given pose has significant contribution from at most a handful of pose clusters. In our implementation, we set α to ten. The optimal value of α will change based on the dataset and should be chosen experimentally. However, because we have defined our pose as handle positions, we expect that handle positions across different characters will remain relatively similar.

Pose Clusters	Reduced Dimensions	Vertex Clusters	Mean Error	Size (MB)
5	5	Uncompressed	0.011	4.5
		10000	0.011	2.0
		5000	0.011	1.5
		1000	0.012	1.0
5	10	Uncompressed	0.008	8.1
		10000	0.008	3.0
		5000	0.008	2.0
		1000	0.009	1.1
5	15	Uncompressed	0.007	11.6
		10000	0.007	4.0
		5000	0.007	2.5
		1000	0.009	1.2
10	5	Uncompressed	0.008	8.8
		10000	0.008	3.9
		5000	0.008	2.8
		1000	0.009	1.9
10	10	Uncompressed	0.006	16.0
		10000	0.007	5.9
		5000	0.007	3.8
		1000	0.008	2.1
10	15	Uncompressed	0.005	23.2
		10000	0.005	7.8
		5000	0.006	4.8
		1000	0.007	2.3
15	5	Uncompressed	0.008	13.1
		10000	0.008	5.8
		5000	0.008	4.1
		1000	0.008	2.7
15	10	Uncompressed	0.006	23.9
		10000	0.006	8.7
		5000	0.006	5.6
		1000	0.007	3.1
15	15	Uncompressed	0.005	34.7
		10000	0.005	11.6
		5000	0.005	7.1
		1000	0.007	3.4

Table 1: Comparison of error and compression based on different parameters. There are 4130 frames in this dataset, composed of walking, running, skipping, reaching, and crouching motions. Error measures are error in the absolute value of ambient occlusion per vertex. For comparison, the method of Kontkanen and Aila results in a size of 10.4MB with a mean error of 0.006.

Our implementation samples and evaluates points only at the vertices of the mesh. These values are interpolated across the mesh faces using the standard graphics driver calls. This works well for our examples because the character is finely tessellated (approximately 67,000 triangles). However, the algorithm does not require that sample points be at the vertices. If a surface parameterization is available, sample points can be located anywhere on the mesh. In such a case, when evaluating the function at a specific pose the method can write to a texture map. The method can then use a fragment shader that reads from this texture map to shade the character.

This paper presents a method for approximating the changes in ambient occlusion due to the motions of the character. Changes in

ambient occlusion can also be caused by the environment. If environmental effects are static, such as the change in ambient occlusion due to a ground plane, they can be taken into account when generating data. The examples in this paper were created using a ground plane. In general, though, environmental effects are not static. However, they also tend to be less important, mainly due to the fact that objects must be relatively close to the character to have a noticeable effect on illumination. Some recent work handles local and non-local effects with different methods, to exploit this fact [Arikan et al. 2005]. The scope of this paper is to handle the changes due to a character's motion, because these tend to be the most apparent. To handle non-static objects, our method should be paired with an algorithm to handle the subtle effects due to the environment. Furthermore, if the effects due to the environment have a low-dimensional parameterization, it may be possible to extend our work to handle those effects.

Our data-driven method is used to build a function for ambient occlusion, which is a complicated function of pose. Accordingly, if the parameters to our approximation are picked poorly, this will result in the failure of our multilinear model. If too few pose clusters are used, or if each pose cluster is projected to too few dimensions, the character skin will develop dark and light blotchy patches. Similar patches appear when rendering a pose that is very dissimilar to those in the training set. The failure mode when using too few spatial clusters is more graceful. The character appears to have large regions of constant color, similar to rendering with a reduced color palette. The solution to each of these failure cases is to add pose clusters, project to a higher number of dimensions, add more poses to the training set, or add more vertex clusters, respectively.

Our method is designed to scale well with large datasets. For each pose cluster, we require a mapping from vertices to spatial clusters, and a coefficient vector for each spatial cluster. Our experiments suggest a high degree of compression is available within a pose cluster. As the size of the training set grows, more pose clusters may be required if different motions are added. However, as we add more data, we expect that the coherence within each pose cluster will increase. Therefore, we expect to get better compression with a larger dataset. In terms of rendering, our method requires a small number of matrix multiplies to project the given pose to each pose cluster space. Within each space, our method performs a low-dimensional dot product per vertex cluster. Therefore, as the complexity of the character increases (*i.e.* more joints), this will only affect the size of the reduced-dimensional projection matrices. Increasing the number of vertices in the character skin will add more complexity only if more vertex clusters are required, then it only adds a coefficient vector per vertex cluster.

An attractive feature of our method is its ability to generate ambient occlusion values for poses not used in the training set, provided these poses are close to those in the training set. We obtained good results when training on approximately one third of the frames in our database. These frames were randomly selected across the set. Furthermore, we were able to generalize the result to different categories of motion. For example, we trained our system on walking and skipping motions, and were able to obtain visually plausible ambient occlusion values when testing on running motion. For evaluation, please see Figure 6 as well as the accompanying video.

Acknowledgments

We would like to thank the Berkeley Graphics Group and the anonymous reviewers for their valuable input. Motion capture data was donated by the Sony Computer Entertainment America Motion Capture Department.

References

ARIKAN, O., FORSYTH, D. A., AND O'BRIEN, J. F. 2005. Fast and detailed approximate global illumination by irradiance decomposition. In *Proceedings of ACM SIGGRAPH 2005*, ACM Press.

ARIKAN, O. 2006. Compression of motion capture databases. In *Proceedings of ACM SIGGRAPH 2006*, ACM Press.

BUNNELL, M. 2005. *Dynamic Ambient Occlusion and Indirect Lighting*. Addison-Wesley Professional, ch. 14.

CHAI, J., AND HODGINS, J. K. 2005. Performance animation from low-dimensional control signals. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3 (August).

CHRISTENSEN, P. H. 2002. Note #35: Ambient occlusion, image-based illumination, and global illumination. *PhotoRealistic RenderMan Application Notes*.

HEGEMAN, K., PREMOŽE, S., ASHIKHMIN, M., AND DRETTAKIS, G. 2006. Approximate ambient occlusion for trees. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 87–92.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (July), 879–887.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005)*.

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

KONTKANEN, J., AND LAINE, S. 2005. Ambient occlusion fields. In *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, ACM Press, 41–48.

LANDIS, H. 2002. Renderman in production. *ACM SIGGRAPH 2002 Course 16*.

LEVIN, D. 1998. The approximation power of moving least-squares. *Mathematics of Computation 67*, 224, 1517–1531.

NG, R., RAMAMOORTHI, R., AND HANRAHAN, P. 2004. Triple product wavelet integrals for all-frequency relighting. In *Proceedings of ACM SIGGRAPH 2004*, ACM Press.

REN, Z., WANG, R., SNYDER, J., ZHOU, K., LIU, X., SUN, B., SLOAN, P.-P., BAO, H., PENG, Q., AND GUO, B. 2006. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. In *Proceedings of ACM SIGGRAPH 2006*, ACM Press.

SATTLER, M., SARLETTE, R., ZACHMANN, G., AND KLEIN, R. 2004. Hardware-accelerated ambient occlusion computation. In *Vision, Modeling, and Visualization 2004*, Akademische Verlagsgesellschaft Aka GmbH, Berlin, B. Girod, M. Magnor, and H.-P. Seidel, Eds., 331–338.

SHEN, C., O'BRIEN, J. F., AND SHEWCHUK, J. R. 2004. Interpolating and approximating implicit surfaces from polygon soup. In *Proceedings of ACM SIGGRAPH 2004*, ACM Press, 896–904.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proceedings of ACM SIGGRAPH 2002*, ACM Press.

SLOAN, P.-P., LUNA, B., AND SNYDER, J. 2005. Local, deformable precomputed radiance transfer. In *Proceedings of ACM SIGGRAPH 2005*, ACM Press.