Real-Time Global Illumination of Deformable Objects

Kayvon Fatahalian
Carnegie Mellon University
Advisor: Doug L. James

May 14th, 2003

Abstract

Existing real-time rendering methods for global illumination use precomputation to simplify runtime computations, but constrain scene geometry to be rigid. This constraint greatly restricts the practical application of these techniques to interactive graphics. This research uses a data-driven approach to support real-time rendering of global illumination effects on deformable surfaces under low-frequency diffuse lighting conditions. The approach is viable in situations where the space of potential object deformations is known offline, so that model appearances can be precomputed for a representative set of deformed states. Principal component data reduction of the precomputed appearance data yields a low-dimensional approximation to the complex appearance space. Runtime interpolation of the reduced appearance basis is then used to efficiently approximate the appearance of other model deformations. Real-time performance is achieved by progressively adapting the rank of the approximation. The method may be implemented on commodity programmable graphics hardware, resulting in only minor runtime cost to the CPU.

1 Introduction

Despite increasingly more powerful graphics hardware, the disparity between the quality of the most realistic computer synthesized images created by offline systems and that of images created by real-time applications remains large. Special effects shots in movies model phenomena so accurately that, to many observers, the synthesized images are often indistinguishable from reality. Yet often even the best images produced by *interactive* graphics applications can immediately be identified as computer generated. One of the primary factors for this discrepancy in image quality is the inability to accurately compute surface illumination due to scene light sources in real time. As a result, real-time graphics applications utilize grossly simplified models of light-surface interaction that are incapable of producing realistic features such as soft shadows due to area light sources or the scattering and interreflection of light in a scene.

Real-time interactive 3D graphics and virtual simulation have proven to be a powerful tool for many purposes, including entertainment, virtual training, and industrial visualization and design. The inability to produce convincing visual images in these applications presents a compelling problem, since accurate rendering is not only important to the visual appeal of such applications, but is often critical to the *utility of the interactive simulation.* When acting within a virtual environment, especially in the case of virtual training exercises involving contact and the precise manipulation of objects, accurate shadowing and surface illumination are important factors in providing a user with the spatial orientation and depth cues that are present in reality. Additionally, realistic light simulation could help engineers identify design flaws in products intended for human use, such as unintended glare in the face of a vehicle operator.

While it is important for interactive simulations to visually mimic reality, it is equally important for the virtual environment to react to forces in realistic ways. Objects modelled as rigid bodies cannot bend or stretch as many real world objects that we manipulate do. Soft objects should deform under contact and respond to forces in a physically correct manner. In medical training applications, tissue should be soft and squishy. Cloth should ripple and sway in response to motion or wind, and rubber or wood

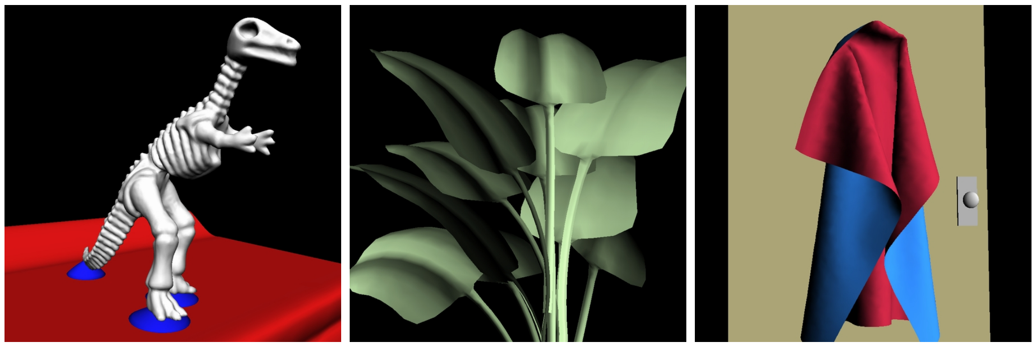

Figure 1: The above images are rendered *without* shadowing or light interreflection effects and illustrate the necessity of more sophisticated illumination models for realistic image synthesis.

surfaces should bend in response to applied pressures. Without modelling such deformation effects, the simulation of user interaction with many types of real world objects is unconvincing.

It is critical that future interactive applications model these deformation responses as accurately as possible and then be able to render the deformable objects in realistic detail. It is obvious that rendering realistic images without plausible motion is of little practical use to interactive simulation. Conversely, realistic dynamic objects must be rendered in a convincing fashion to increase the realism of the simulation.

In reaction to this problem, this thesis seeks to relax a serious constraint of existing real-time global illumination methods, namely, the inability to efficiently render non-rigid scene objects.

1.1 Related Work

1.1.1 Real-Time Rendering of Global Illumination

Real-time rendering systems approximate scene lighting with a small number of unoccluded point sources. Light incident on a surface is assumed to come directly from a point source, while in the real world, non-local scene geometry reflects, occludes, and scatters light, resulting in radiance reaching a surface from many directions regardless of the size and position of scene light sources (global illumination). The generation of realistic images requires the simulation of global illumination effects. For the most part, increasingly powerful graphics hardware has allowed for more detailed geometric models, higher resolution textures, and vastly improved frame rates, but has not added abilities to more accurately compute complex lighting behavior. Standard offline techniques accounting for realistic the transport of light through a scene and the sampling of the entire incident light field at surface points, such as radiosity [2] and Monte Carlo path tracing [3], are too costly to be viable real-time approaches for dynamic scenes. Recent work has shown that ray tracing can be performed at high speed on programmable graphics hardware [16], but generating soft shadows and global illumination effects from this approach remains too costly for the real time applications.

The lack of shadowing effects in real-time rendering led to hardware multi-texturing techniques where surface illumination was computed accurately offline, and then encoded into a texture map [18] (or per-vertex color) used to modulate the surface color obtained from a second traditional texture map. This approach dictates that lighting and geometry rendered using these light maps be fixed in the interactive environment. Dynamic multi-pass rendering techniques designed to work within the graphics hardware pipeline have been used to render shadows. Shadow mapping [21][18] makes clever use of the hardware z-buffer to estimate depth and occlusion information in a scene in order to generate dynamic shadows, but the approach yields poor quality shadows unless extremely high-resolution shadow maps are used. Shadow volumes [4] produce better results and account for object self-shadowing, but are inefficient even in simple cases, since a large amount of "shadow" geometry must be added to the existing geometry of the scene. Additionally, neither of these real-time shadowing techniques generalizes efficiently to generating soft shadows due to area lights.

Light field rendering techniques [11] attempt to circumvent the deficiencies of simplified real-time rendering models by measuring the appearance of a real world object from many different locations in space, and then utilizing the acquired data to generate new images of the object from arbitrary views at runtime. Since the acquired images capture the complex lighting conditions and surface properties present in the real world, the synthesized views account for these features as well. However image-based rendering approaches require the sampled data to be sufficiently dense to represent the entire set of desired runtime view angles and object positions. Sampling this space to sufficient detail results in a enormous number of source images that must be processed at runtime. Recent research has focused on compressing the required runtime data [1] to enable hardware accelerated approaches.

The spherical harmonic (SH) functions have been shown to be an efficient basis for representing arbitrary reflectance properties of a surface in progressive radiosity simulations [19] and for the real-time representation of incident hemispherical light fields [17]. Sloan et al. [20][9] achieved real-time performance in rendering objects with global illumination effects by using the spherical harmonics as a basis for "*radiance transfer functions*" that capture how the object occludes and scatters incident light onto itself and neighboring surfaces. The approach utilizes a preprocessing step to determine how light is transported as a result of interaction with object geometry. As a result, run time computation is reduced to the application of the geometry-dependent precomputed transfer functions to a spherical harmonic projected light field. The radiance transfer method is capable of rendering objects in real time under arbitrary low frequency lighting conditions and is well tailored for implementation on modern programmable graphics hardware. Although simple dynamic scenes are possible, the approach relies on an offline precomputation step to determine surface transfer functions, and therefore is limited to only rigid objects.

1.1.2 Real-Time Deformation

The accurate simulation of physical deformation using numerical techniques, such as finite element simulation, can be prohibitively costly for use in the real-time domain. As a result, real-time deformation approaches have relied on data-driven techniques to generate reasonable approximations to physical deformation. Shape morphing techniques [13] succeed in producing small scale deformations by interpolating between a large set of geometric poses created either by an artist or obtained via offline physical simulation.

Pose space deformation [12] improved upon commonly used skinning methods by combining skeletal subspace deformation [14] with shape morphing, and parameterized the deviation of the deformed surface from a skeleton driven base position by a set of parameters in an abstract pose space.

James and Pai [7] performed offline modal analysis of elastic models to determine the displacement of model vertices due to a small set of principal deformation modes. These displacements were then linearly combined using programmable graphics hardware to efficiently render object deformation at runtime. To ensure accurate surface appearance, per-vertex normal corrections were computed at runtime via the linear combination of precomputed basis perturbations. Eigenskin [10] generalized the approach presented in [7] by utilizing offline finite element simulation to determine skin surface deformation corresponding to various joint configurations. Runtime deformation was parameterized in a compact basis for the precomputed space of deformations obtained via a mixture of principal component analyses [15]. The data reduction step allowed runtime computation of geometric deformation to be performed on programmable graphics hardware with negligible main CPU cost, yielding high frame rates while approximating the results obtained from the original offline simulation to a high degree of accuracy.

1.2 Motivation and Questions

Existing real-time global illumination techniques exploit extensive amounts precomputation to achieve performance goals. As a result, the geometry of objects must remain fixed at runtime, preventing the rendering of deformable objects with such techniques. With prior knowledge of potential object deformations, a naive solution is to perform the preprocessing step on all potential geometric configurations, and use the appropriate precomputed data at runtime for rendering. Obviously, this approach is intractable due to both time and space limitations. Yet data-driven approaches have shown the effectiveness of using selective offline precomputation to create convincing approximate results for arbitrary real-time deformations. Similar techniques may be employed to utilize selective precomputation of model appearance information to facilitate the approximate computation of the global illumination of a dynamic object in various states. This thesis investigates whether this is a viable approach to rendering a dynamic, deformable object with global illumination effects at real-time frame rates by generalizing the existing method of *precomputed radiance transfer* to scenes containing deformable objects. This investigation deals with the following principal questions:

1. *Can the interpolation of precomputed appearance information be employed to obtain acceptable visual results?*

   Due to the occlusion, reflection, and scattering of light, appearance information is global in scope. A small change in an object's geometry may result in significantly different lighting conditions elsewhere (e.g. a door cracked open slightly allowing light to stream into a room). The complexity of lighting conditions may make surface appearance information a poor candidate for interpolation.

2. *How much precomputation is required to obtain sufficient detail about the space of possible appearances so that interpolated results will adequately approximate actual appearances?*

   In complex scenes, computing global illumination information is an very expensive task that can consume hours of CPU time. Although precomputation is desired to accelerate runtime performance, we wish to minimize preprocessing times to maximize the usefulness of the approach.

3. *How can the technique be adapted to fit within the resource limitations of commodity graphics hardware?*

   Precomputation may yield massive amounts of data, however, in order to achieve acceptable performance, the size of the data set used at runtime must be able to fit in system memory, or ideally, the limited amount of the high performance memory contained on the graphics board of a PC. If such compression can be obtained, computations can be performed entirely on the programmable graphics board, freeing CPU cycles for other tasks.

2 Global Illumination via Real-Time Radiance Transfer

We elected to build upon the radiance transfer rendering technique as opposed to other real-time methods for several reasons. Interactive environments demand maximal runtime rendering flexibility, and radiance transfer allows for objects to be illuminated by arbitrary low-frequency light sources as well as to move rigidly in relation to other scene geometry. Secondly, although we constrain our approach to diffuse surfaces, radiance transfer generalizes to surfaces with arbitrary surface BRDFs [9], such as glossy or anisotropic materials. Finally, programmable graphics hardware accelerated implementations of radiance transfer are possible.

Precomputed radiance transfer as first presented by Sloan et al. [20] is described briefly in this section.

For diffuse surfaces, L'_P, the reflected radiance from surface point P is given by the integral [8]:

$$L'_P = \frac{\rho}{\pi} \int \max(N \cdot s, 0) L_P(s) ds$$

Where ρ is surface reflectance and N the surface normal at point P. L_P is a scalar function defined over the unit sphere of directions ($s = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$) representing the light radiance incident on surface point P. L_P may vary across the surface due to scattering and shadowing properties of neighboring geometry.

Assuming infinitely distant or nearly spatially invariant lighting and no occlusion or scattering effects, then the incident radiance on an *object* is given by $L(s)$. In this case, for all P, $L_P(s) = L(s)$. However, another point on the object's surface (or on the surface of another object), P', may occlude the distant light source from P in the direction s (figure 2). In this case, $L_P(s) \neq L(s)$. Rather, $L_P(s)$ equals the radiance reflected from P' in the direction of point P. The difference between $L_P(s)$ and $L(s)$ is entirely dependent upon the geometry and material properties of the objects in question.

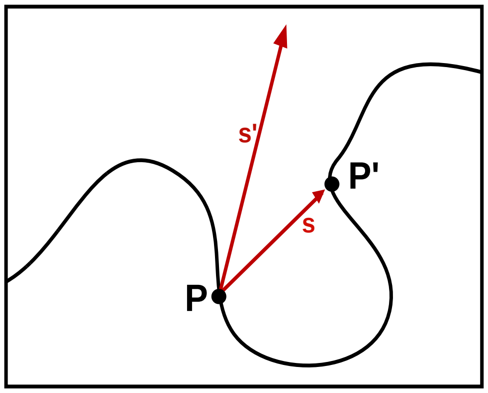

Figure 2: Sampling incident radiance at surface point P: P is occluded by P' in the direction s, and thus receives reflected radiance $L^{'}_{P'}(-s)$ from this direction. Direction s' in not occluded, so P receives radiance $L(s')$ from the direction s'.

Radiance transfer uses an offline preprocessing step to compute a set of functions (the radiance transfer field) defined at points on an object's surface such that the lighting integral can be factored into an entirely object dependent term, $T_P(s)$, and a light dependent term that is constant across the surface of the object. [1]

$$L^{'}_P = \int T_P(s)L(s)ds \tag{1}$$

T_P contains information about the self-shadowing and light interreflection properties of the object. T_P specifies how incident radiance on the *object* is transferred into exit radiance at a particular surface point P.

By projecting the scalar functions $L(s)$ and $T_P(s)$ into the basis of spherical harmonic functions, computation of the integral in (1) is simplified to the dot product of vectors consisting of the projection coefficients of $L(s)$ and T_P. Retaining only coefficients from the first n spherical harmonic bands yields n^2-length vectors $(T_P)_i$ and $(L)_i$ that represent approximations to the functions $T_P(s)$ and $L(s)$ in the spherical harmonic basis. $L^{'}_P$ is then given by:

$$L^{'}_P \approx \sum_{i=1}^{n^2} (T_P)_i (L)_i \tag{2}$$

[1]The reader is referred to appendix A for details regarding our particular implementation of the radiance transfer preprocess, and how the functions T_P are determined and evaluated numerically.

Since transfer vectors $(T_P)_i$ are determined for a given model during the offline preprocessing step. Runtime calculations consist only of projecting the light field $L(s)$ into its spherical harmonic coefficients and computing the dot product of the lighting vector with the transfer vector at each sampled point P on the model. Light transport (performed offline) as well as light integration (reduced to dot product operation) computations do not need to be performed in real time.

In practice, transfer functions are calculated at a finite number (s) of surface points on an object so transfer information for the entire model is represented by the object's radiance transfer vector, T.

$$T = \left[\begin{array}{cccc} T_0 & T_1 & ... & T_{s-1} \end{array} \right]^T$$

In our implementation 4th-order spherical harmonics $(n = 4)$ are used, so $(T_P)_i$ is a 16-component vector. Radiance transfer vectors are computed computed for each rgb color channel, so that the transfer vector for the entire object is of length $48s$.

High frame rates are possible when rendering objects with radiance transfer since the dot product between the light and transfer vectors for an object can be performed in graphics hardware in a vertex program. If transfer vectors components are encoded in texture maps, fragment program implementations are also possible.

3 Precomputing Appearance Data For Deformable Objects

3.1 Terminology

A deformable object may enter a large number of geometric configurations based on the type and magnitude of deformation it undergoes in an interactive environment. A single geometric configuration is referred to as a *pose*, u, existing in the *deformation space*, $\mathcal{U}$ of the object. In our implementation, each pose is represented by a vector of per-vertex displacements from a mean pose, and $\mathcal{U}$ is approximately spanned by the columns of deformation basis matrix U. The coordinates of any particular pose are given by deformation state vector q_u.

Changes in object geometry alter the light transport properties of the surface. Therefore the transfer vector T is a function of the deformation state, $T = T(q_u)$. Radiance transfer data is referred to as *appearance data* since it determines how the rendered object looks. The set of all appearances is referred to as the model's appearance space, $\mathcal{A}$, and is approximately spanned by the columns of appearance basis matrix A. Each model appearance is specified by appearance state vector $q_a = q_a(q_u)$. Thus, the state q of a deformable object is then given by coordinates in both deformation and appearance space.

$$q = \left[\begin{array}{c} q_u \\ q_a \end{array} \right]$$

3.2 Sampling the Appearance Space

It is assumed that information about the object's deformation space is available during precomputation. This restriction is not extremely severe, since data-driven approaches to interactive deformation also require offline knowledge of the deformation space. Although the space of possible deformations is known, it may be highly complex, making it is implausible to precompute the corresponding transfer fields for all points in deformation space. A small, but representative sampling of appearance space must be precomputed, and the values for arbitrary appearances approximated via the interpolation of the precomputed appearance state vectors. It is desirable to sample the appearance space by precomputing appearance data for as many poses as possible, however, precomputation of the transfer field for a static object is an expensive process that may take hours of CPU time for complex models (see results in section 6). Therefore, the amount of available computation time dictates the number, N_a, of poses that can be sampled. Ideally, the N_a poses should be chosen so that the corresponding precomputed appearances form a good sampling of the appearance space.

3.2.1 Clustered Sampling

Attempting to capture maximal information about the appearance space with a sparse number of samples, we select a set of N_a poses uniformly distributed in deformation space. Deformation space is partitioned into N_a regions via K-means clustering [15]. A state q_u^k located at the mean of the k^{th} cluster $(0 \leq k < N_a)$ is selected and radiance transfer preprocessing is performed on the associated pose geometry to obtain the corresponding appearance vector T^k associated with appearance coordinate q_a^k.

$$q_a^k = q_a(q_u^k) \tag{3}$$

The space spanned by the appearance vectors T^k approximates $\mathcal{A}$. Only the appearance data for the poses q_u^k is known, so interpolation of the known states q_a^k is used to approximate the appearance of any arbitrary pose q_a^j. Normalized radial basis functions [15] are used to interpolate the scattered known appearances,

$$q_a^j = q_a(q_u^j) = \sum_k \hat{\phi}_k(q_u^j) c_a^k \quad k = 1 \ldots N_a. \tag{4}$$

The form of the basis functions $\hat{\phi}_k$ is given in Appendix B. The values c_a^k are determined by substituting the data from (3). We construct a $48s$-by-N_a matrix A whose columns are the precomputed appearance vectors $(range(A) \approx \mathcal{A})$.

$$A = \begin{bmatrix} | & | & & | \\ T^0 & T^1 & \ldots & T^{N_a - 1} \\ | & | & & | \end{bmatrix}$$

Thus the interpolated appearance vector T^j is given by the matrix-vector product:

$$T^j = A q_a^j \tag{5}$$

4 Appearance Space Data Reduction

Although sparse sampling of the appearance space reduces runtime appearance computations to the linear combination of a set of N_a basis vectors, this simple approach remains insufficient for real-time performance. If the object deformation is non-trivial, N_a must be allowed to be reasonably large to ensure that the appearance space is sufficiently sampled. 50 poses were precomputed to sample the space of the deformable dinosaur model (see section 6), which was the sparsest sampling of the three examples of our technique. In addition, due to the dense sampling of transfer functions on the object's surface, the basis transfer vectors are large. Each dinosaur transfer vector consumes 5.2MB of storage, so in excess of 250MB of data must be loaded at runtime and interpolated each frame for appearance synthesis alone.

Not only can such a large amount of data not be processed at real-time frame rates, the size of the data set prevents it from fitting entirely into the high speed memory available on graphics hardware. Thus, a graphics hardware implementation is not possible.

Generating a more sparse sampling of transfer functions (decreasing s) over the surface of the object is a poor method of reducing data, since reflected radiance depends on the direction of the surface normal in relation to that of incoming light. Decreasing the sampling frequency of transfer functions along the object's surface would result in a significant loss of appearance detail when rendering intricate surfaces.

It is preferable to exploit redundancies in the appearance data to obtain a set of $\tilde{N}_a$ $(\tilde{N}_a \ll N_a)$ appearance vectors that approximately span the range of the T^k's.

4.1 Data Reduction via Principal Component Analysis

Principal component analysis is used to obtain a basis for an N-dimensional data set such that variation of the data is maximized along the coordinate axes. Data reduction is achieved by ignoring the axes of lowest variation, thereby projecting the original data into a lower dimensional space with minimal loss of accuracy. In figure 3, a set of data points is located in 2D space. The data exhibits equal variation along the x and y axis, however in the primed coordinate system, there is significant variation along the x' axis and little along y'. Thus, the points can be represented in a one-dimensional space (along the x' axis) while still retaining much of the information present in the two-dimensional representation.

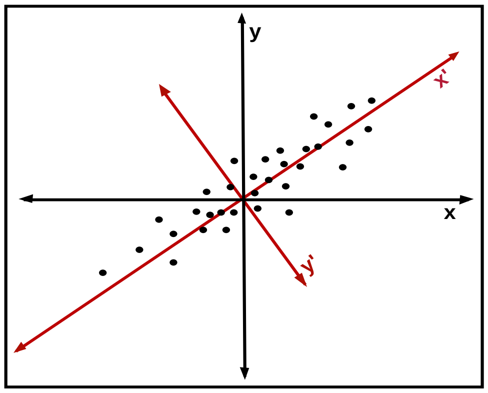

Figure 3: Variation of the data is greatest along the orthogonal axes of the primed coordinate system.

We compute of the mean, T_{mean} of the N_a basis appearance vectors and form matrix A' whose columns $\tilde{T}^k$ are obtained by subtracting the mean from each of the T^k's ($T^k = T_{mean} + \tilde{T}^k$).

Principal component analysis is performed via Singular Value Decomposition (SVD) [5] on the matrix A'. SVD factors $A^{'}$ into an orthonormal $48s$-by-N_a matrix Y whose columns span the range of A', a diagonal N_a-by-N_a matrix S, and a N_a-by-N_a matrix V^T.

$$A' = YSV^T = \begin{bmatrix} | & | & & | \\ \tilde{T}^1 & \tilde{T}^2 & \dots & \tilde{T}^{N_a} \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_{11} & & & \\ & \sigma_{22} & & \\ & & \ddots & \\ & & & \sigma_{N_a N_a} \end{bmatrix} V^T$$

The diagonal elements σ_{kk} of S are the singular values of A', and σ_{kk} gives the variation of the data along the axis specified by $\tilde{T}^k$. We normalize the σ_{kk}'s such that the largest singular value is 1, and form matrix $\tilde{A}$ whose columns consist of $\tilde{T}^k$ whose corresponding singular value is greater than ε. The resulting columns of $\tilde{A}$ form a basis for a low rank approximation to the range of A' that is accurate to ε% least squares error.

$$span(\tilde{A}) \approx span(A')$$

Let $\tilde{N}_a$ be the rank of $\tilde{A}$. Now, the appearance of an arbitrary model deformation is represented by reduced space coordinate $\tilde{q}_a^j$. $\tilde{q}_a^j$ consists of $\tilde{N}_a$ components of $SV^T q_a^j$ corresponding to basis vectors retained in $\tilde{A}$.

The appearance vector $\tilde{T}^j$ is then computed by:

$$\tilde{T}^j = T_{mean} + \tilde{A}_r \tilde{q}_a^j \tag{6}$$

Runtime computation is reduced to the linear combination of $\tilde{N}_a$ basis vectors added to the mean model appearance. In practice, only a few basis vectors are required to approximate the original appearance space with reasonable accuracy ($\tilde{N}_a \ll N_a$), resulting in a significant reduction in the size of the required runtime data set.

SVD does not require that T_{mean} be subtracted from the precomputed appearance vectors. However, we do perform this step to avoid the wasteful computation of $\tilde{T}_0(\tilde{q}_a)_0)$ in the computation in (6), since without removal of the mean, there is little variation in the magnitude of the first principal component for various model states.

In additional, removal of the mean prior to performing SVD reduces the range of the resulting singular values, and thus the variation of magnitude of the various coefficients in $\tilde{q}_a$. Reducing this variation is important to the potential implementation of the algorithm using a fixed-point representation on graphics hardware. Widely ranging values would cause significant loss of precision when data is scaled to fit the range of fixed point formats.

5 Runtime Implementation on Graphics Hardware

We wish to exploit features of modern graphics processors, namely their highly parallel stream processing architecture and single-clock dot product instruction between 4-component floating point vectors to efficiently perform the computations required by (6) each frame. Additionally, delegating the large matrix-vector multiplication to the graphics processor frees the main CPU to perform other tasks (such as computing object deformation) required by the interactive application.

Our hardware implementation is targeted at the feature set of the of NVIDIA CineFX line of processors, since the chips exhibit desirable features such as 32-bit floating point precision throughout the entire hardware pipeline, and the ability to perform a large number of instructions per vertex or fragment program execution. It is important to note at this time that we are working with pre-release versions of the graphics board and drivers, and have encountered serious performance problems with our preferred hardware rendering approach. Despite communication with NVIDIA engineers, we have not yet succeeded in resolving these issues. As a result, the performance numbers provided later in this thesis reflect an *alternate hardware/software* hybrid implementation of the necessary runtime computations that achieves real-time frame rates but places certain restrictions on scene lighting. It is believed that our initial entirely graphics hardware accelerated approach remains a viable method, and that in time, as driver and hardware issues are resolved by NVIDIA, a technique similar to that described below will yield real-time frame rates at *negligible main CPU cost.* Both rendering approaches are discussed in detail in this section.

5.1 Graphics Hardware Rendering Approach

In order to determine the reflected radiance (color) at each sampled surface point, we require access to 48 values in each of the $\tilde{N}_a$ basis appearance vectors (16-component vectors per color channel representing the projection of the surface point's transfer function into the 4^{th}-order SH basis). Modern graphics processors limit the number of per-vertex values accessible to a vertex program to 64, so we must perform our computations in the fragment processing stage of the graphics pipeline, where we have access to a virtually unlimited amount of data stored in the form of textures. Additionally, it is *preferred* that computation be performed in the fragment processing stage of the pipeline so that the data needed for appearance interpolation does not consume valuable vertex program resources required for the hardware accelerated computation of the object's geometric deformation [10][6][7].

5.1.1 Overview

Runtime rendering in OpenGL is performed via a two-pass method. The first pass performs the computations specified by (6) to generate $\tilde{T}^j$ as well as computes the the dot product between the components of $\tilde{T}^j$ and the lighting vector to obtain the reflected radiance at each sampled surface point. These colors are written to a texture that is used to modulate the surface color of the deformable model in a second rendering pass.

The basis appearance vectors are stored in high performance graphics memory at application start, thus, only the 48-component lighting vector and the $\tilde{N}_a$ component appearance state vector must be transferred from system memory to the graphics card each frame.

We encode the $\tilde{T}^k$'s and T_{mean} into $\tilde{N}_a + 1$ rectangular 2D texture maps. Recall that the rgb transfer vector contains 48 components per sampled surface point. Since each texel holds 4 values, 12 contiguous texels are required to store the SH projection coefficients for any surface point.

5.1.2 The Two-Pass Rendering Process

At the start of the first pass, the view is set to an orthographic projection and OpenGL state is set such that rendering writes pixels to an off-screen buffer. The buffer is sized such that each rendered pixel contains the color of one of the s sampled surface points on the deformable model. We render a single quad positioned to exactly cover the entire viewport and set texture coordinates for the $\tilde{N}_a + 1$ textures so that each block of 12 contiguous texels corresponds exactly to a single screen buffer pixel.

Next, the lighting and current appearance state vectors are stored in constant memory accessible to a Cg fragment program. [2] The fragment program is executed once per rendered pixel (per sampled surface point), and performs the following operations per execution:

1. The program receives hardware interpolated 2D texture coordinates (u, v) which are set by the OpenGL application such that for each rendered pixel, (u, v) specifies precisely the starting location of the block of texels containing coefficients representing the transfer vector at the corresponding model surface point. 48 values are read from each of the $\tilde{N}_a$ textures as well as from the mean appearance texture using a total of $12(\tilde{N}_a + 1)$ texture fetch operations.

2. The approximate interpolated transfer vector is computed using the formula given in (6).

3. The dot product of the interpolated transfer vector and lighting vector is computed (4 dot product operations per color channel), yielding the rgb color value for the model surface point.

4. The resulting color is written to the frame buffer.

When the first pass is complete, the frame buffer is set as the source image of the active texture map. The view and OpenGL state are restored to the settings required by the application and the deformable object is rendered in traditional fashion using only the texture map to determine the color values of object surface points (*OpenGL lighting is disabled*).

5.1.3 Rendering Performance Issues

Even when using very small bases ($\tilde{N}_a < 5$) We were unable to obtain acceptable real-time performance (< 10 fps) using the steps described above. The numerous texture fetch operations performed in the first phase of our method brought runtime performance to a halt. Tests showed that performance was limited by the number of texture fetches, since the frame rate decreased sharply as the number of fetch operations increased, but did not diminish if only additional mathematical operations were added to the fragment program. We first attempted to store the appearance basis vectors as 128 bits-per-texel floating point textures. Hypothesizing that video memory bandwidth limitations were the cause of our performance problems we switched to a lower precision 32 bits-per-texel fixed-point texture format. Surprisingly, reducing the amount of data obtained per texture fetch *did not* yield significantly better results.

[2] Our Cg source code for the first pass fragment program (for the case where $\tilde{N}_a = 4$) is compiled into a 317 instruction fragment program by the NVIDIA Cg compiler.

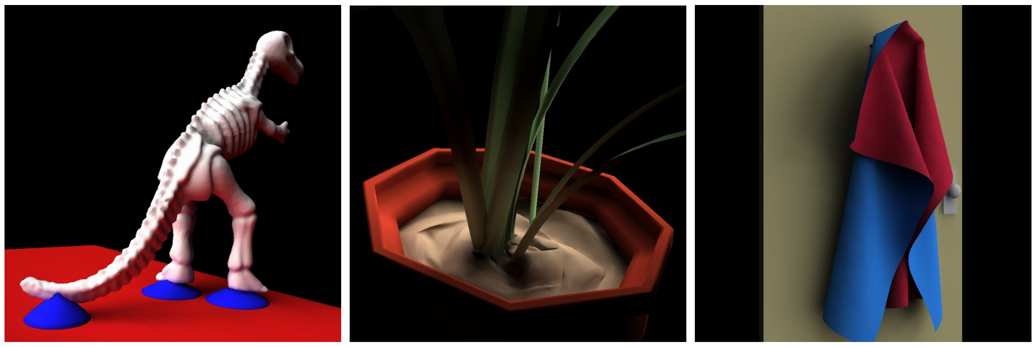

Figure 4: The deformable dinosaur, plant, and cloth models are rendered with soft shadowing and diffuse interreflection effects due to area light sources present in the scene.

5.2 Hybrid Software/Hardware Rendering Approach

In response to hardware implementation difficulties, we successfully implemented a method that distributes computation between the CPU and GPU. In this approach we achieve high frame rates over intervals where the light sources remain static, and allow for lower frame rates at times where the light field changes between frames.

The dot product between the lighting vector and the appearance basis vectors is computed *in software*, resulting in $\tilde{N}_a$ $3s$-length vectors C^k specifying the surface color at each sampled surface point. Now, only the linear combination of $\tilde{N}_a$ 3-vectors is required to compute the interpolated color of a given surface point (as opposed to length-48 transfer vectors). Since our preprocessing step samples surface transfer functions at the model's vertices, we assign the $\tilde{N}_a$ basis colors as varying parameters to a vertex program that interpolates the values and outputs the result as the color value for that vertex.

The values for each of the C^k are stored as OpenGL vertex array ranges in video memory to avoid data transfer over the AGP bus each frame. As long as the incident light field remains unchanged, only the appearance state vector $\tilde{q}_a$ is transferred to the card each frame.

When scene light sources are altered, the dot product of the lighting and basis appearance vectors are re-evaluated in software and the new C^k's transferred into video memory. As a result, performance decreases significantly during intervals where the scene lighting changes from frame to frame.

In addition to the restriction on lighting changes, our partial software implementation is less than ideal since resources are consumed in the vertex-processing stage of the hardware pipeline to perform color interpolation calculations. Recall that in our application, model deformation is performed via the linear combination of per-vertex displacements (corrections) to a mean geometric model. Typically, the deformation space basis displacements are stored as per-vertex attributes, and deformed vertex positions are computed efficiently in hardware using a vertex program. However, for interesting scenes, the 64 values of per-vertex memory are insufficient to hold both the displacement and appearance bases. As a result, basis displacements that do not fit in per-vertex memory are linearly combined and added to the mean pose each frame in software. Performance decreases since the perturbed mean position for each vertex must be transferred to the card each frame.

6 Results

The real-time rendering of deformable objects with global illumination effects is demonstrated in three example applications:

1. **An elastic dinosaur** attached to a surface by suction cups located on the dinosaur's feet. The interactive simulation allows for impulses to be applied to the dinosaur in forward, left, right, up, and down directions.

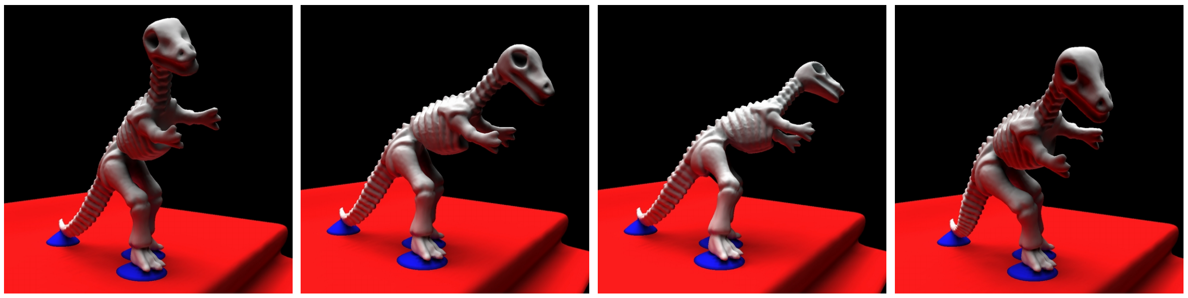

Figure 5: The dinosaur bends as it is forced in various directions. Note how the side of the entirely white dinosaur's ribcage receives red light diffusely reflected from the ground plane in all but the third image, where the dinosaur's deformation orients its side away from the ground and correctly receives no red reflected light

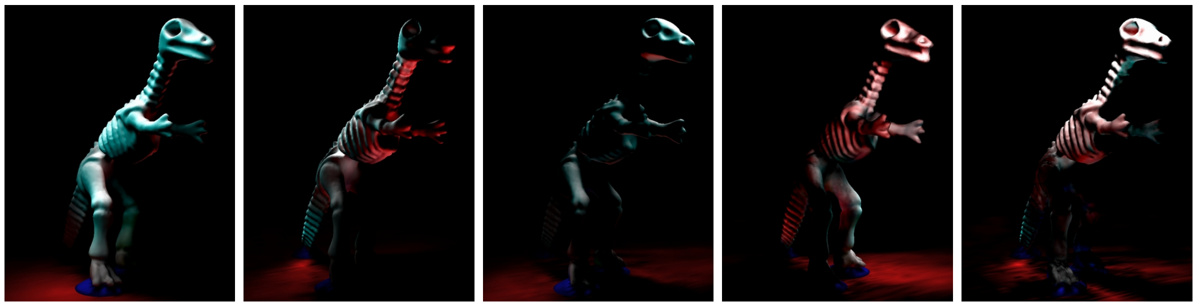

Figure 6: The first 5 dinosaur reduced appearance basis vectors (corrections to the mean). Correction magnitudes are magnified to increase visual clarity.

2. **Cloth hanging from a door hanger** that responds to the opening and closing movement of the door. The cloth features many folds that create complex self-shadowing situations.

3. **A potted plant** whose leaves shake and sway in response to movement of the pot to the left or the right. Upper leaves cast shadows onto the lower leaves and lower leaves scatter incident light upward onto the back side of the upper leaves.

In each of these examples, the dynamic model's interactive response to external impulses is computed by the method given in [6].

6.1 Radiance Transfer Preprocessing Times

Surface transfer functions accounting for both self-shadowing and diffuse interreflection effects we computed. Preprocessing was performed on a 2.0Ghz Intel Xeon machine with 2GB of 266MHz DDR SDRAM. Preprocessing times for the three example scenes are given in Table 1.

Table 1: Scene Preprocessing Statistics

Scene	Triangles	Surface Samples (s)	Poses (N_a)	Time per Pose	Total Time
Dinosaur	55360	27839 (2000 dirs)[3]	50	88.8 min	74 hrs
Cloth	25742	16569 (1500 dirs)	200	21.4 min	71 hrs 27 min
Plant	11155	9989 (1250 dirs)	100	11.6 min	11 hrs 40 min

6.2 Data Compression

Before dimensionality reduction via principal component analysis, storing the appearance space basis vectors consumes a significant amount of storage. As shown by Table 2, the datasets are too large to fit in system memory of a common PC, and far exceed the video memory limitations of commodity graphics hardware.

Table 2: Basis Memory Requirements (Before Reduction)

Scene	Transfer Size	Poses (N_a)	Basis Data
Dinosaur	5.1 MB	50	254.8 MB
Cloth	3.0 MB	200	606.8 MB
Plant	1.8 MB	100	182.9 MB

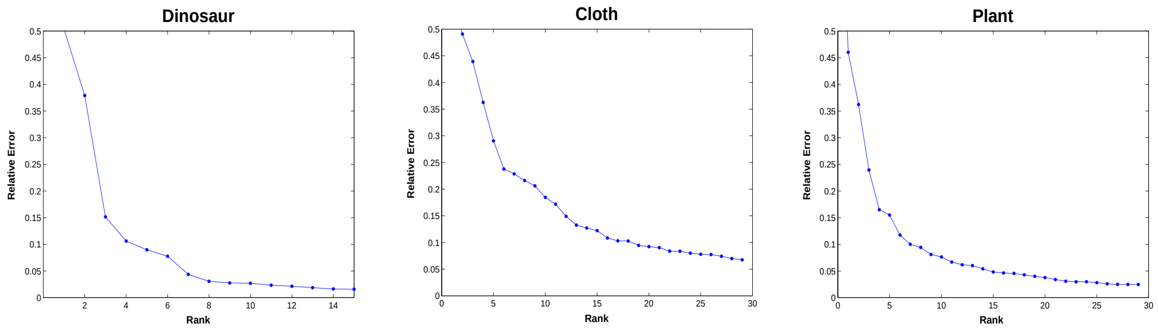

Figure 7: Relative error in approximation as a function of reduced basis rank (N_a).

However, data reduction was successful in producing a visually acceptable approximation to the original data while only consuming a small fraction of the original memory space (see figure 7). In the case of the dinosaur, only 3 principal components (94% data reduction) are necessary to approximate the original sampled data to 15% error. Lighting conditions in the plant and cloth scenes are more complex, thus approximation using a small number of principal components introduces slightly more error than in the case of the dinosaur. Examples of the cloth and plant models rendered with varying appearance rank are provided in figures 8 and 9.

6.3 Runtime Performance

Figure 10 shows rendering performance as a function of the rank of the reduced dinosaur appearance basis. Note that the frame rates specified in all figures and tables referenced in this section pertain to intervals over which scene lighting is fixed.

In the final interactive demos, in addition to appearance interpolation, the computation of the model's deformation response to applied impulses [6] must be computed each frame. Table 6.3 gives runtime performance both when computing the interpolated appearance and displacement vectors entirely in software and when splitting the computation between the CPU and the GPU as discussed in section 5.2.

The sw/hw performance of the dinosaur demo is notable since both the interpolated displacement and appearance vectors are computed entirely in hardware (bases fit in vertex program memory). In this case, our hardware/software implementation significantly outperformed the pure software implementation.

However, contrary to expectations, in the plant and cloth examples, sw/hw performance was not significantly better that of the software only implementation. In these cases, the size of the reduced

[3]A set of random directions on the unit sphere is used to numerically sample the transfer functions during the preprocess (see Appendix A).

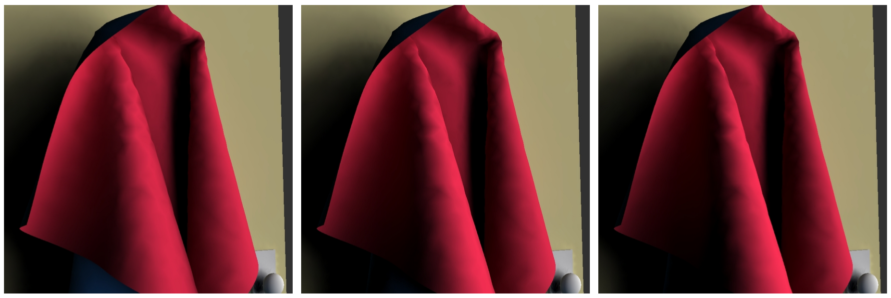

Figure 8: Increasing the number of reduced appearance basis vectors used to render the cloth model increases the sharpness of shadows. The mean appearance is given on the left, and renderings using 3 and 6 corrections to the mean are given at center an on the right.

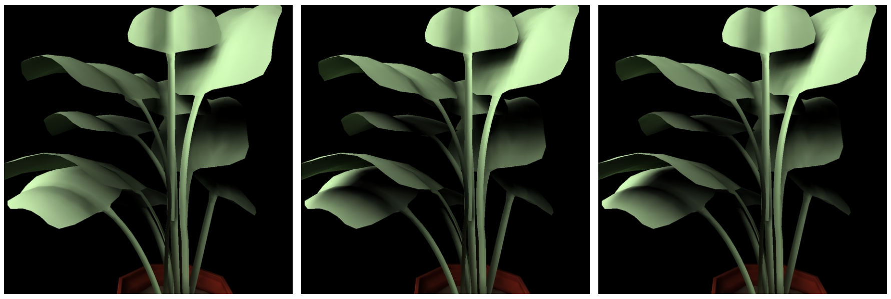

Figure 9: Increasing the number of reduced appearance basis vectors used to render the plant adds shadowing detail that is not captured using lower rank approximations. The mean plant appearance is shown at left. The center and right images were created using 3 and 6 corrections respectively.

displacement and appearance bases required a fraction of the interpolation to be performed in software. Even if the GPU performed a significant amount of the required computations, the sw/hw implementation yielded only marginally better performance. We attribute this result to the high cost of sending results computed in software over the system bus to the graphics processor each frame. Any performance gained by performing computation on the GPU is offset by the cost of moving data to the GPU.

7 Conclusions and Future Work

Precomputing the surface transfer functions for various deformation states of a model, and then extracting the principal components of the resulting data yields a reduced basis for the space of model appearances. Appearance coordinates in this basis may be interpolated in real time for the synthesis of approximate appearance information for any model pose.

Results indicate that the error introduced by sparsely sampling the appearance space, performing dimensionality reduction on the data, and then interpolating between a small set of basis states can yield visually acceptable results. The technique produces convincing visual results even in complex nonlocal shadowing conditions such those created by light filtering through the leaves in our plant example.

While a graphics hardware implementation is highly desirable, the results show that real-time performance can even be achieved by performing appearance interpolation in software on a PC. Although the method is capable of running in software, a critical objective of future work is to perform runtime appearance interpolation entirely on the GPU, ensuring that the method can be performed at negligible main CPU cost.

Future work should look to increase the accuracy of the reduced appearance basis. It is notable that in the cloth scene, expanding the reduced basis from 20 to 30 vectors only improves the approximation by

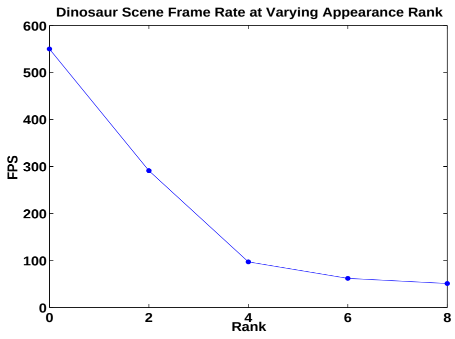

Figure 10: Dinosaur runtime performance decreases as the size of the reduced appearance basis increases. Performance was measured on a 2.6GHz P4 with a clocked down NVIDIA QuadroFX pre-release graphics board

Scene	Displ. Basis Size	Appearance Basis Size ($\tilde{N}_a$)	FPS (sw only)	FPS (sw/hw)
Dinosaur	11	6	82	175
Cloth	30	12	149	158
Plant	18	12	200	(unavailable)

Table 3: **Results**: Runtime performance depends upon both the size of the displacement and reduced appearance bases. Performance was measured on a 2.0Ghz Intel Xeon machine with 2GB of 266MHz DDR SDRAM, and a fully clocked NVIDIA QuadroFX graphics board.

about 3%. Numerous folds, self contact, and the cloth's close proximity to the door create an appearance space characterized by localized high frequency details (largely due to shadowing effects). These features are not well suited for extraction by principal component analysis, which identifies the global features of a data set. Other data analysis methods may be better suited for obtaining a reduced basis that approximates the sampled appearance space to greater accuracy.

However, closely approximating the sampled appearance space does not guarantee acceptable visual results if significant detail is missed during the sampling process. It is possible that although the precomputed poses are chosen to be well distributed in deformation space, the corresponding appearances may not form an adequate sampling of all potential appearances, leaving regions of the appearance space unsampled. In such cases visual accuracy will be poor when attempting to interpolate appearances in the undersampled region. *Importance sampling* should be used to select poses from $\mathcal{U}$ whose corresponding appearances exhibit significant variation and form a good sampling of $\mathcal{A}$.

Lastly, further reducing the size of required runtime data is necessary for this technique to be useful in the context of an interactive simulation consisting of many objects as well as for increasing the runtime

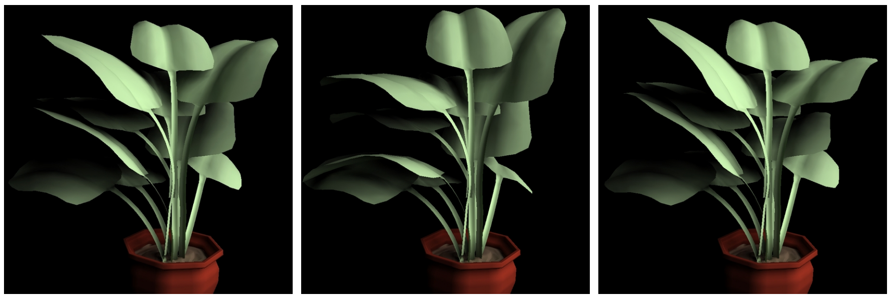

Figure 11: The plant's leaves are jiggled as the potted plant is moved to the left and the right.

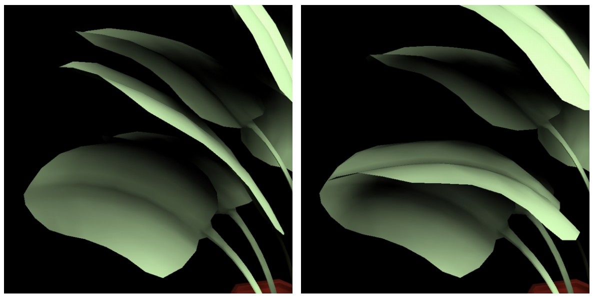

Figure 12: Two plant poses show the dynamic movement of shadows as the two leftmost leaves bend towards each other.

performance of our method. Piecewise PCA analysis of the transfer fields on patches of the model may yield sets of bases that better approximate the appearance data over certain regions of the model. A more accurate approximation would allow fewer basis vectors to be used to achieve equivalent visual results.

8 Acknowledgements

I would like to thank Doug L. James for advising (and working) on this project. Thanks also goes out to Matt Trentacoste for advice and assistance in working with programmable graphics hardware and to Joel Welling for providing octree code used in the radiance transfer preprocess. This project made use of hardware donated by NVIDIA.

A Radiance Transfer

The explanations in this appendix come largely from Sloan et al. [20].

By observing that the illumination of surface point P is due to unoccluded light from the distant light field $L(s)$ and the light $L^{'}_{P'}$ reaching P after reflection from other surface points P', (1) can be rewritten as:

$$L^{'}_{P} = \frac{\rho}{\pi} \int \Big(H_P(s) V_P(s) L(s) + H_P(s)(1 - V_P(s)) L^{'}_{P'}(s) \Big)\, ds \tag{7}$$

Where $H_P(s)$ is just $\max(N \cdot s)$ from (1). $V_P(s)$ is defined to be 1 if a surface point P' occludes P from the distant light field, and 0 otherwise. $L^{'}_{P'}(s)$ is the radiance reflected from P' towards P in the direction s.

Since lighting is assumed to be *nearly spatially invariant*, the light at P' is that as that at P. $L^{'}_{P'}(s)$ is a function of only $L(s)$, thus, $L^{'}_{P} = L^{'}_{P}(L(s))$, and $L(s)$ can be factored out of both terms in (7) to give (1) repeated below:

$$L^{'}_{P} = \int T_P(s) L(s) ds$$

A.1 Numerical Computation of Surface Transfer Functions

Our preprocess implementation follows the method described by Sloan et al. [20]. Surface transfer functions that account for both shadowing and diffuse interreflection effects are computed numerically via a multi-pass ray-casting preprocess. First, a random set of N_s directions S is generated, and the values of the n^2 SH functions $y_i(s)$ are computed and cached for each direction $s \in S$. In a the first pass, shadow rays are shot into the scene to determine, at each vertex, the directions s that are occluded by other surface geometry. Transferred radiance is determined by projecting $\frac{\rho}{\pi}H_P(s)$ into the SH basis using numerical integration over the directions s.

$$(T_P)_i = \frac{1}{N_s}\sum_s \frac{\rho}{\pi}H_P(s)V_P(s)y_i(s)$$

Note that energy is not added from occluded directions (shadowing).

In ensuing passes, interreflections are added by adding energy only from *occluded directions.* Reflected radiance from other surface points P' in pass p is obtained from the computed value in the previous pass. (superscript denotes pass number)

$$(T_P)_i^p = (T_P)_i^{p-1} + \frac{1}{N_s}\sum_s \frac{\rho}{\pi}H_P(s)(1-V_P)(T_{P'})_i^{p-1}(s)y_i(s)$$

A.2 Additional Implementation Notes

- Shadow ray-surface intersection tests take time logarithmic in the number of model triangles, so the runtime cost of the preprocess is $\Theta(\text{numPasses} \cdot \text{numSurfaceSamplePts} \cdot N_s \cdot lg(\text{numTriangles}))$.
- Surface transfer vectors are computed at each of the model's vertices. For double-sided surfaces, such as the cloth mesh or the leaves of the plant, two transfer vectors are computed per vertex, one using the given vertex normal and the second using the inverted normal.
- Sloan et al. integrate over 10-30K random directions, however in our case, since many model poses must be preprocessed, far fewer samples (between 1250 and 2000) were used. The coarser sampling did not introduce noticeable noise in the renderings, except in the case of the cloth scene, where the close proximity of surfaces required a highly sampling of the unit sphere to accurately capture occlusions.
- We precomputed transfer for various poses of a single model in parallel on several machines to reduce preprocessing times. However, our sparse random sampling of the sphere of directions introduced noise in the precomputed data, resulting in noticeable variability in surface reflectance across poses computed using different sets of random direction samples. To solve this problem, ensured that the *same set* of directions was used during the preprocessing of every pose of a given deformable model.

B Normalized Radial Basis Functions

Normalized radial basis functions are used to interpolate $q_a(q_u)$ for all q_u. We center radial basis functions at each of the N_a precomputed poses,

$$\phi_k(q_u) = e^{-\|q_u^k - q_u\|^2/a^2}, \quad k = 1\ldots N_a, \tag{8}$$

where a is the radial scale of the basis functions. Then the q_a are interpolated by:

$$q_a(q_u) = \sum_k \hat{\phi}_k(q_u)c_a^k \tag{9}$$

where c_a^k are the unknown coefficients, and $\hat{\phi}_k(q_u)$ are normalized basis functions,

$$\hat{\phi}_k(q_u) = \frac{\phi_k(q_u)}{\sum_i \phi_i(q_u)}.$$

The components of c_a^k can be determined by substituting the known appearance coordinates $q_a^k = q_a(q_u)$, to obtain (4) repeated below.

$$q_a^j = q_a(q_u^j) = \sum_k \hat{\phi}_k(q_u^j) c_a^k \quad k = 1 \ldots N_a.$$

References

[1] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk. Light field mapping: efficient representation and hardware rendering of surface light fields. *ACM Transactions on Graphics (TOG)*, 21(3):447–456, 2002.

[2] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, Boston, MA, 1993.

[3] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, 1984.

[4] Cass Everitt and Mark J. Kilgard. Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering. 2002.

[5] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.).* Johns Hopkins University Press, 1996.

[6] Doug L. James and Kayvon Fatahalian. Precomputing Interactive Dynamic Deformable Scenes. *ACM Transactions on Graphics (TOG)*, 2003.

[7] Doug L. James and Dinesh K. Pai. DyRT: dynamic response textures for real time deformation simulation with graphics hardware. *ACM Transactions on Graphics (TOG)*, 21(3):582–585, 2002.

[8] James T. Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150. ACM Press, 1986.

[9] Jan Kautz, Peter-Pike Sloan, and John Snyder. Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics. In *Proceedings of the 13th workshop on Rendering*, pages 291–296. Eurographics Association, 2002.

[10] Paul G. Kry, Doug L. James, and Dinesh K. Pai. EigenSkin: real time large deformation character skinning in hardware. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 153–159. ACM Press, 2002.

[11] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM Press, 1996.

[12] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172. ACM Press/Addison-Wesley Publishing Co., 2000.

[13] George Maestri. *Digital Character Animation 2: Essential Techniques.* Pearson Education, 1999.

[14] N. Magnenat-Thalmann, R. Laperriere, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33. Canadian Information Processing Society, 1988.

[15] Oliver Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. Springer Verlag, 2000.

[16] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 703–712. ACM Press, 2002.

[17] Ravi Ramamoorthi and Pat Hanrahan. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 497–500. ACM Press, 2001.

[18] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252. ACM Press, 1992.

[19] Francis X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. *ACM SIGGRAPH Computer Graphics*, 25(4):187–196, 1991.

[20] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics (TOG)*, 21(3):527–536, 2002.

[21] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.