Key Point Subspace Acceleration and Soft Caching

Mark Meyer John Anderson
Pixar Technical Memo #06-04b
Pixar Animation Studios

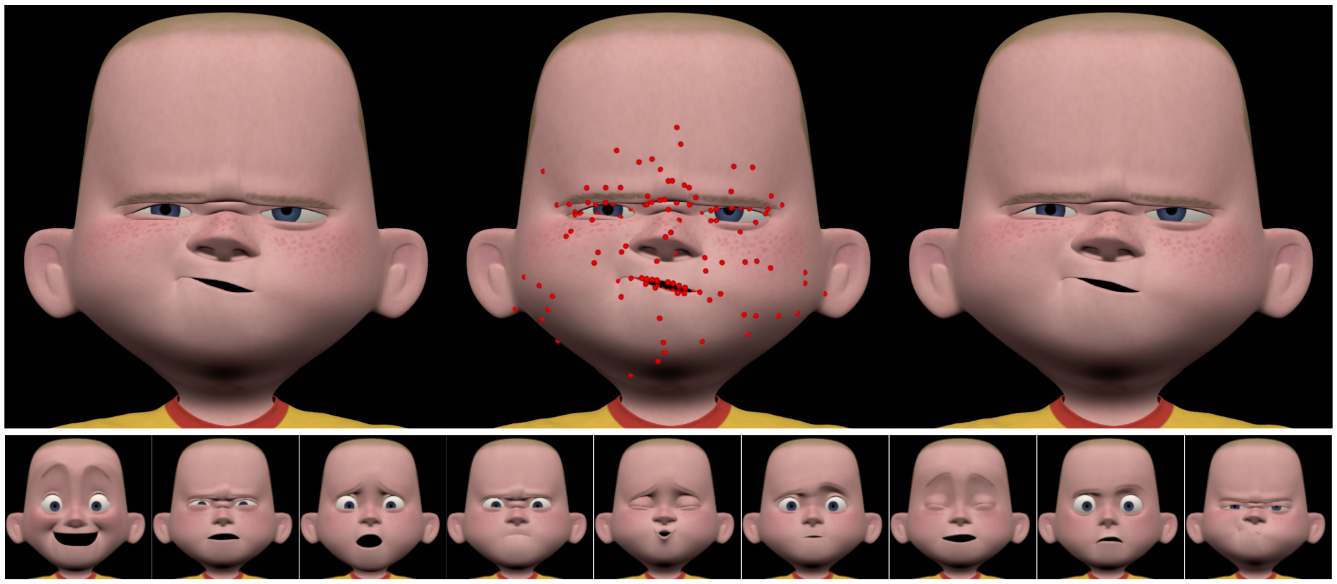

Figure 1: *An example of the Key Point Subspace Acceleration (KPSA) technique applied to character animation: (Top) Left: The facial model computed by posing all 2986 articulated facial points using our in house posing engine. Middle: The 170 key points computed by our KPSA algorithm. Right: The facial model computed by posing only the 170 key points using our in house posing engine and then using KPSA to generate the 2986 facial points. Using the KPSA technique results in an 8.7x speedup in posing time. (Bottom) Several different poses generated by our KPSA algorithm are displayed to demonstrate the range of poses possible using the KPSA technique.*

Abstract

Many applications in Computer Graphics contain computationally expensive calculations. These calculations are often performed at many points to produce a full solution, even though the subspace of reasonable solutions may be of a relatively low dimension. The calculation of facial articulation and rendering of scenes with global illumination are two example applications that require these sort of computations. In this paper, we present Key Point Subspace Acceleration and Soft Caching, a technique for accelerating these types of computations.

Key Point Subspace Acceleration (KPSA) is a statistical acceleration scheme that uses examples to compute a statistical subspace and a set of characteristic key points. The full calculation is then computed only at these key points and these points are used to provide a subspace based estimate of the entire calculation. The soft caching process is an extension to the KPSA technique where the key points are also used to provide a confidence estimate for the KPSA result. In cases with high anticipated error the calculation will then "fail through" to a full evaluation of all points (a cache miss), while frames with low error can use the accelerated statistical evaluation (a cache hit).

Keywords: Animation, Subspace Analysis, Statistical Models

1 Introduction

Many applications in graphics require expensive calculations to be performed at many different locations. Facial articulation and rendering global illumination are two such example applications - facial articulation computing the deformation of each point on the face, and rendering computing the global illumination at many different locations in the scene. Although these computations are performed at many different locations, there is often correlation between the solution values at these locations - they are not independent. Due to this fact, recently there has been a great deal of interest in the use of statistical models / subspace methods for many calculations in graphics; a sampling of which is described below.

Subspace methods have shown to be promising for character articulation and deformation tasks. One line of research illustrated by Lewis [2000] and Wang and Phillips [2002] involves the creation of a kinematic articulation model that is trained from poses which can be generated from either physical simulations or hand corrected posed models. The Lewis approach is based on a pose based interpolation scheme in animation variables while Wang and Phillips base their approximation on multiple coefficient weighting of the positions of points in various skeletal articulation frames. Both of these approaches are more general than our technique since they do not require the posing of key points, a limitation that precludes the use of models depending upon history based simulation for posing.

The weakness of the kinematic schemes is related to their generality - the required training sets for these methods can be very large and it is essentially impossible to place bounds on the errors of the reconstructions when poses are seen that are far from those included in the training set.

The advent of programmable graphics hardware has enabled the development of several subspace based deformation techniques that run in real time with minimal main CPU costs [James and Pai 2002; James and Fatahalian 2003; Kry et al. 2002]. EigenSkin [Kry et al. 2002] is one such technique for real time deformation of character skin. EigenSkin uses principal component analysis to compute a basis for the skin displacements resulting from perturbing each joint of a model. Given several example poses consisting of joint angles and skin displacements, specific joint angle values can be associated with coordinates in the basis space by projecting the skin displacement into the basis. Skin displacements for novel joint angle configurations are computed by using the joint angles to interpolate these example basis coordinates, and the resulting coordinates determine a displacement represented in the subspace formed by the basis.

Subspace methods have also been beneficial in Rendering and Illumination computations as shown by research in Precomputed Radiance Transfer [Sloan et al. 2003] and its variants such as Precomputed Local Radiance Transfer [Kristensen et al. 2005]. Precomputed Local Radiance Transfer (PLRT) is a method for accelerating the computation of illumination in a scene. First, the illumination is computed for several example configurations (light positions, etc.) the resulting illumination is stored at each vertex as the coefficients to a spherical harmonic basis. Clustered PCA is then performed on these coefficients to create a subspace and the subspace coordinates (projection onto the basis vectors of the subspace) of the example configurations are computed. The illumination from a novel lighting position can be computed by using the light position to interpolate the subspace coordinates of nearby example lights, and the resulting subspace coordinates determine the illumination represented in the subspace.

Subspace techniques are also often used in machine learning and computer vision applications such as Facial Recognition/Reconstruction [Hwang et al. 2000], Facial Inpainting [Mo et al. 2004], and Expression Mapping [Zhang et al. 2003]. Zhang *et al.* [2003] use a subspace method to synthesize facial expressions from a small set of tracked feature points on a performer's face. These feature points are chosen by hand using knowledge about a performer's face. The subspace method allows the systhesized character to have much greater detail then the small number of tracked feature points would produce on their own.

Our Key Point Subspace Acceleration (KPSA) scheme is similar to other subspace techniques in that it computes a subspace from examples and uses this subspace model to accelerate the corresponding calculation. A key component of our technique is that instead of using animation variables such as joint angles and light positions to drive the subspace model, we instead use the samples of the calculation itself (skin displacements, illumination, etc.) at a set of carefully chosen key points to drive the projection. It is the use of these key points that gives our subspace based acceleration technique many of its unique properties and advantages as described in the remainder of the paper.

Contributions

The contribution of this paper is a general acceleration/caching scheme with the following important properties:

- **Generality** – the technique does not use domain specific knowledge and is applicable to several different applications (facial articulation, global illumination, etc.). Additionally, the acceleration/cache structure is keyed on the output of the calculation at a small set of key points, which means that the cache is independent of the parameterization of the input variables.
- **Automatic Key Point Selection** – The key points are automatically selected (again, without domain specific knowledge). The key points are chosen such that the behavior of the key points describes all the important nonlinearities of the problem.
- **Soft Caching** – by computing an error estimate using the key point values, the subspace acceleration scheme can be used selectively - like a statistical or soft cache. In cases with high estimated error, a "cache miss" occurs causing the full calculation to be performed at all points, while in cases with low estimated error, the subspace estimate can be used. This cache like behavior allows the subspace scheme to be used more widely than using a subspace scheme directly.

Taken together, this results in a powerful acceleration technique for certain Computer Graphics applications.

2 KPSA and Soft Caching

This section describes the KPSA and Soft Caching (SC) technique in more detail. Although KPSA-SC is a general acceleration scheme, for clarity and concreteness, the text and examples given in this section will often refer to the example application of facial articulation. A second application is shown in section 3.

A schematic block diagram of the general statistical modeling problem is shown in Figure 2. In this general model we have a core black box that we would like to train to accept the animation variables (posing controls, light positions and properties) and produce the desired outputs (point positions or illumination values). The problem with this most general form is that the relationships between the input controls and the outputs can be highly nonlinear - for example it is common to use an articulation variable to set either the sensitivity or pivot point of another variable - an extremely nonlinear process that is essentially impossible to discover statistically from a training set.

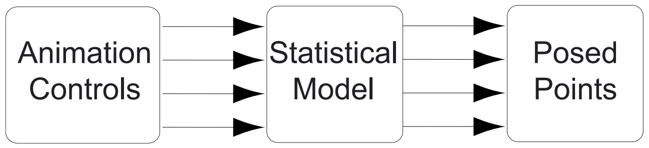

Figure 2: *Block Diagram of a basic statistical modeling system for animation: The trained statistical model accepts as input the animation variables and produces the posed points as output. Unfortunately, since the relationships between the animation variables and the posed points can be extremely nonlinear, both the statistical model as well as the training set must account for these complex nonlinearities.*

In order to deal with these nonlinearities, the KPSA approach is structured slightly differently as shown in Figure 3. In a preprocess, a training set is used to create a statistical subspace model as well

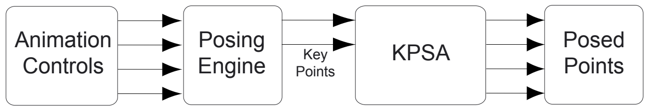

Figure 3: *Block Diagram of the KPSA system applied to animation: Rather than taking the animation controls directly as input, KPSA instead computes the positions of a small set of key points using the standard posing engine and then takes these posed key points as input. KPSA then uses the values of these key points to produce all of the posed points. The advantage here is that if the key points are chosen such that the behavior of these key points describes all of the important nonlinearities of the problem, the statistical model can compute the remaining point values without having to deal with these nonlinearities.*

as a set of key points. At runtime, the full computation is performed only at these key points. The computed values are then projected onto the subspace (via least squares projection) and the resulting pose space coordinates are used to generate values for all points (including the key points).

More formally, we wish to approximate a nonlinear function $\mathbf{f}$ using a linear subspace:

$$\mathbf{f} \approx \hat{\mathbf{f}} = \mathbf{A}\mathbf{p} \tag{1}$$

where $\mathbf{f}$ is the function to approximate (represented as a column vector: $\mathbf{f}_i$ is the displacement of point i for articulation, or the color of point i for rendering), $\hat{\mathbf{f}}$ is the corresponding subspace approximation, $\mathbf{A}$ is the subspace matrix of size $numPoints \times numBasisVecs$ (whose columns $\mathbf{a}_i$ are the basis vectors), and $\mathbf{p}$ are the subspace coordinates.

At runtime, the subspace coordinates are computed by first calculating the function $\mathbf{f}$ at a small subset of key points producing $\mathbf{f}_{key}$. These values are then projected onto the subspace to minimize the error $||\mathbf{f}_{key} - \hat{\mathbf{f}}_{key}||$:

$$\mathbf{p} = \arg\min_{\mathbf{p}} ||\mathbf{f}_{key} - \mathbf{A}_{key}\mathbf{p}|| \tag{2}$$

where $\mathbf{f}_{key}$, $\hat{\mathbf{f}}_{key}$ and $\mathbf{A}_{key}$ are versions of $\mathbf{f}$, $\hat{\mathbf{f}}$ and $\mathbf{A}$ that contain only the rows corresponding to the key points. Note that by using values computed at the key points to drive the subspace model, we can avoid having to discover and model the nonlinear interaction of the animation variables and use a much simpler and more robust model for the remaining points.

Since we have chosen to use a model that is linear in the key point values, we should choose our key points from the training set such that the behavior of the key points describes all of the important nonlinearities of the problem. It is of course quite possible that there may be new nonlinearities that are not seen in the training set and that is where the need for the soft caching extension arises. When using the soft caching extension, an error estimate due to the subspace approximation is computed using the key points. If the error estimate is high, the cache misses and a full computation is performed at all points. When the error estimate is low, the values are computed using the more efficient subspace model.

In the remainder of this section we will discuss the key issues: how we select the subspace and key points, and how we provide error estimates to be used in the fail through process.

2.1 Building the Subspace

In order to build the subspace, we start with a training matrix $\mathbf{T}$ whose columns $\mathbf{t}_i$ correspond to example calculations or "poses" of $\mathbf{f}$. This matrix therefore has dimensions $numPoints \times numPoses$. We then use *Principal Component Analysis* (PCA) to determine the values for the subspace basis vectors $\mathbf{a}_i$. We use only the M most significant basis vectors to produce our M-dimensional subspace $\mathbf{A}$. M is chosen such that the subspace projection error of the training set is acceptably small.

It should also be noted that once the final number of basis vectors is chosen, $numBasisVectors = M$, the $\mathbf{a}_i$ vectors are not uniquely determined by the minimization of error. The only critical property is the subspace spanned by the basis vectors. In fact the multiplication by any nonsingular matrix will result in vectors that span the same space and the multiplication by any orthogonal M dimensional rotation matrix will result in an orthonormal basis for the subspace. This property is quite useful and is frequently exploited in the factor analysis community [Harmon 1976] to generate basis vectors that are more "local" in some sense than the original basis vectors. We will take advantage of this when we select the key points.

2.2 Selection of Key Points

When using the KPSA technique there are two potential sources of error. The first of these is the projection error that results from the fact that the target pose itself may not be in the subspace:

$$\begin{aligned} e_{proj} &= ||\mathbf{f} - \mathbf{A}\,\mathbf{p}_{proj}|| && (3)\\ \mathbf{p}_{proj} &= \arg\min_{\mathbf{p}} ||\mathbf{f} - \mathbf{A}\,\mathbf{p}|| && (4) \end{aligned}$$

The second is the cueing error - the error that results from the fact that the subspace location $\mathbf{p}$ determined from the least squares fit to the key points (Equation 2) may not be the closest point in the subspace to the desired pose:

$$e_{cue} = ||\mathbf{A}\,\mathbf{p}_{proj} - \mathbf{A}\,\mathbf{p}|| \tag{5}$$

Cueing error is due to using an inadequate set of key point values as distance proxies. We have derived an iterative approach for selecting the key points that attempts to minimize this cueing error. Although there are many different techniques for choosing key points (e.g. [Shashua and Wolf 2004]), our technique is relatively simple and has worked well in our applications.

There are often critical fiducial points identified by the character (or lighting) designer. We initialize our key points to contain these points. For rendering applications we may also add a sparse grid of points to ensure that no large regions are devoid of key points.

Our approach then generates a block of basis vectors, typically 10 to 20. The basis vectors often have large support with multiple maxima which makes directly examining the raw basis vectors to choose the key points problematic. For example, Figure 4 (top) shows a few of the basis vectors computed from an articulation application. Notice how the basis vectors represent global motions of the face.

Although it is difficult to use the raw basis vectors to choose the key points we can transform them to make them more useful. As mentioned in the previous section, we can apply a rotation to the basis vectors without changing the spanned subspace. We use the Varimax method [Harmon 1976] which computes an orthogonal rotation that maximizes the ratio of the 4th moment to the 2nd moment of the basis vectors, a measure of locality. Intuitively, this

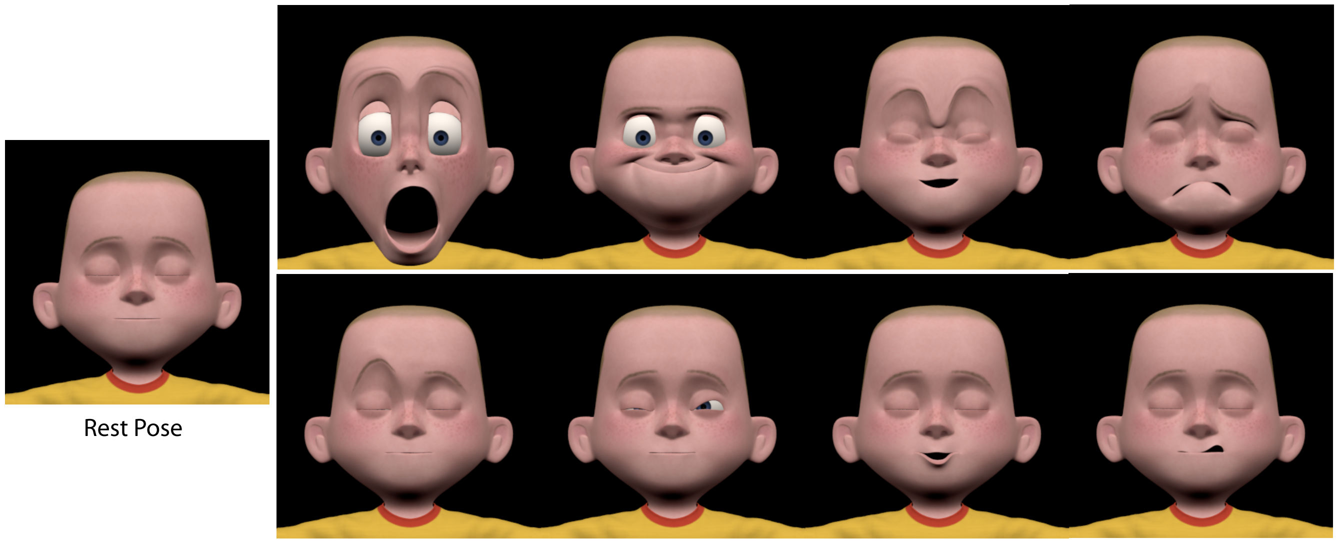

Figure 4: *An example of some of the basis motions computed from a facial articulation application. (Top) Raw basis vectors computed by PCA. Notice how the motion is quite global. (Bottom) Basis vectors after Varimax rotation. Notice how the motions are much more localized leading to easier selection of representative points.*

rotation localizes the basis vectors by maximizing the variation of a small set of points in each basis vector (in turn driving the variation of the other points towards zero). Figure 4 (bottom) shows a few of the basis vectors after the Varimax rotation. Notice how the motions are much more localized making it easier to determine representative points for each of the motions.

Once we have a set of suitably conditioned basis vectors we choose two points from each basis vector $\mathbf{v}$, the first point (i) is the point with the largest magnitude in the basis vector and the second (j) is the point whose inner product with our first point has the largest negative value:

$$i = \arg\max_k \|\mathbf{v}_k\| \quad (6)$$

$$j = \arg\min_k (\mathbf{v}_i \cdot \mathbf{v}_k) \quad (7)$$

This type of technique is used in many fields to discover statistical structure and is known as teleconnection analysis. This technique is essentially choosing the point that moves the most when this (localized) basis function is excited as well as the most negatively correlated point. For example the rotated basis vectors shown in Figure 4 (bottom) would produce key points on the brow, eyelid and lips.

Given this set of key points and the subspace basis vectors, we can compute a KPSA approximation for each of the columns of the training matrix $\mathbf{T}$. Subtracting these approximations from the corresponding columns of $\mathbf{T}$ creates a residual matrix that represents the error (both projection and cueing error) when approximating the training set using the current set of key points. We then repeat the key point selection process to add additional key points, using the residual matrix to compute the basis vectors. Using the residual in this iterative process allows the key point selection to choose points to reduce the cueing error.

The process terminates when the maximum residuals reach an acceptable error bound. A subtle but important point is that the basis vectors constructed during this phase in the second and later batches should now be discarded - all basis vectors used for the runtime KPSA reconstruction should be computed from the initial training matrix $\mathbf{T}$.

The process of selecting key points can be summarized as follows:

```
keyPoints = initial user defined key points
$\mathbf{resid} = \mathbf{T}$
Iterate until resid is small:
    Compute basis vectors from resid
    Varimax rotate the basis vectors
    Choose the control points from each rotated basis vector (excluding points that
        are already in keyPoints) and add them to keyPoints
    Reconstruct the columns of $\mathbf{T}$ using KPSA with
        the current set of key points, call this $\hat{\mathbf{T}}$
    $\mathbf{resid} = \mathbf{T} - \hat{\mathbf{T}}$
```

2.3 Soft Caching and Fail Through

Although estimates produced by the KPSA technique are usually quite accurate, there can be times when the statistical reconstruction is unable to produce an estimate with acceptable accuracy. An example of this is shown in Figure 5 for the application of facial animation. The facial pose resulting from posing all of the points with our posing engine is shown on the left, while the result generated via KPSA using 170 key points is shown in the middle. Notice how the tight lip pucker is incorrectly reconstructed.

Since we compute the function values at the key point positions ($\mathbf{f}_{key}$), we can use these to compute the root mean square key point error:

$$e_{key} = rms(\mathbf{f}_{key} - \hat{\mathbf{f}}_{key}) \quad (8)$$

Although this error is biased, if the key points are well chosen, the error in this calculation is often a good measure of the total projection error ($e = rms(\mathbf{f} - \hat{\mathbf{f}})$). Large projection errors occur in cases when we are calculating the results for a frame where an animation control is exercised that was not exercised in the training set or a particular combination of animation controls is exercised that result in a novel pose.

This suggests the possibility that we could treat the KPSA result like a cache and use a large key point projection error as an indication of a “cache miss”. Since the domain of applicability of our method involves calculations easily performed on a point by point basis, in the event of a cache miss, we could simply go on to

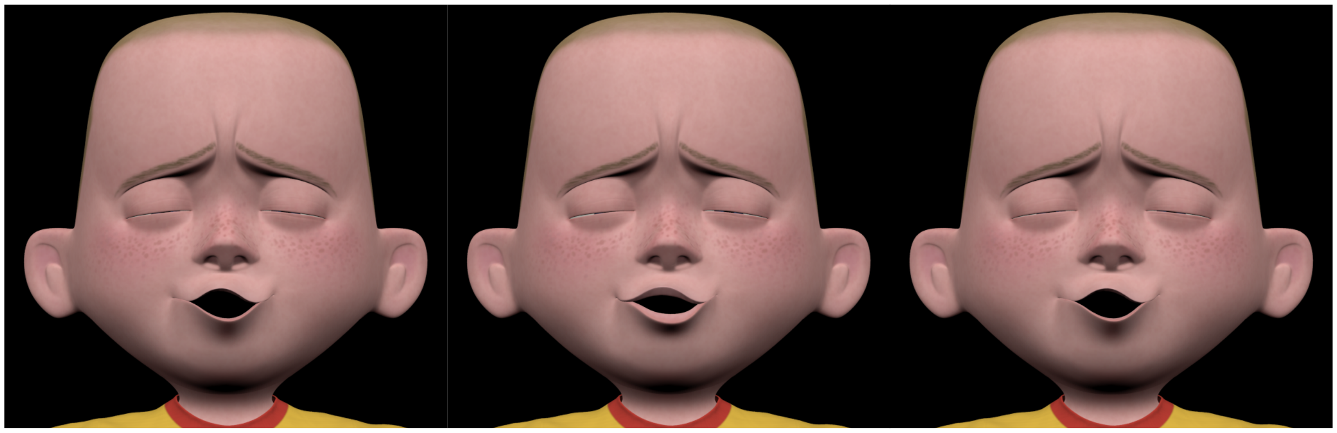

Figure 5: *An example of Soft Caching applied to character animation: (left) The face generated by posing all 2986 points using our in house posing engine. (middle) The pose generated by KPSA using 170 key points. Notice how the lips are improperly reconstructed. (right) That pose generated using Soft Caching with $\varepsilon = [0.1, 0.15]$. Notice that the pose generated by KPSA was correctly identified as inaccurate using the key point error and the cache miss fail through correctly interpolated the KPSA computed pose with the fully computed pose producing much more accurate lips.*

perform the full calculation on all of the points. For animated sequences somewhat more subtlety is required since we do not want to have any discontinuous behavior that will lead to popping. To eliminate this problem we incorporate a transition zone of projection error values $\varepsilon = [\varepsilon_{min}, \varepsilon_{max}]$ over which we do perform the full calculation but interpolate between the KPSA computed result and the full calculation result as a function of error:

$$if(e_{key} < \varepsilon_{min}): \quad output = \hat{\mathbf{f}} \tag{9}$$

$$if(e_{key} > \varepsilon_{max}): \quad output = \mathbf{f} \tag{10}$$

$$otherwise: \quad output = \alpha\mathbf{f} + (1-\alpha)\hat{\mathbf{f}} \tag{11}$$

where $\alpha = \frac{e_{key} - \varepsilon_{min}}{\varepsilon_{max} - \varepsilon_{min}}$.

Figure 5(right) shows the result of applying Soft Caching to the incorrectly reconstructed facial pose in Figure 5(middle) using $\varepsilon = [0.1, 0.15]$. Notice how the cache miss was correctly identified and the resulting pose is much more accurate. We should also mention that poses identified as cache misses can be tagged and used as additional training for the KPSA-SC system. This results in a system that learns more about the pose space as it is used. Training could be done each night resulting in a more accurate KPSA-SC system for the animators and lighters to use the next day.

2.4 The Complete KPSA-SC Algorithm

Now that we have described all of the pieces, the complete KPSA-SC algorithm can be formulated as follows:

```
Preprocess
    1) Compute the basis vectors $\mathbf{a}_i$ from the training set $\mathbf{T}$ (Section 2.1)
    2) Compute the key points (Section 2.2)
Runtime
    1) Compute values only at the key points: $\mathbf{f}_{key}$
    2) Least squares project the key points onto the subspace
        to find the subspace coordinates $\mathbf{p}$ (Equation 2)
    3) Reconstruct all points using $\mathbf{p}$ (Equation 1):
        $\hat{\mathbf{f}} = \mathbf{A}\mathbf{p}$
    4) Soft Cache (Section 2.3):
        Compute $e_{key} = rms(\mathbf{f}_{key} - \hat{\mathbf{f}}_{key})$
        Using $e_{key}$ and the user defined $\varepsilon$, compute the soft cache output,
        blending $\mathbf{f}$ and $\hat{\mathbf{f}}$ if necessary
```

Note that the runtime component of the KPSA algorithm is extremely cheap (compared to the full solution), only requiring the computation of the key point values, a small least squares projection and a linear combination of the basis vectors.

3 Results

This section discusses the results of applying our KPSA technique to two different application domains: Character Articulation and Rendering Global Illumination.

3.1 Applications to Character Articulation

One application of the KPSA technique is the computation of non-skeletal character articulation, particularly facial articulation. Faces are particularly difficult to replace with statistical models because they often have hundreds of animation controls, many of which are very complex and non-linear. On the other hand the results of Guenter *et al.*'s [98] work have shown that, at least for human faces, the number of real degrees of freedom is often quite small. This suggests that a subspace based technique should work well for faces. Unfortunately, using the animation variables directly to drive the subspace model is difficult due to the complex nonlinearities. Additionally, faces do not have the benefit of skeletal structure and joint angles used by other parts of the body as input to their statistical models. By using the computed key point positions as input, our KPSA technique is able to avoid modeling these complex non-linearities and performs well when applied to facial articulation.

In order to test our technique we used the facial animation for a boy character from an entire full-length CG feature film. This is a particularly difficult character for several reasons. First, the boy is extremely animated and produces many exagerated facial poses. Second, the boy is a superhero with super speed and this results in extremely large deformations as well as extreme anticipation and follow through.

The data set for the boy consists of 8322 frames of animation from 137 animation sequences. In order to provide an independent data test for the KPSA process we divided the animation sequences into two sets, one was used to train the model (2503 poses chosen from 44 randomly selected sequences) and the second was used for validation (the remaining 5819 poses). We found that 85 basis vectors posed by 170 key points was sufficient to pose the model to very small errors in the training segments. The character with the key points indicated is shown in the top-middle of Figure 1.

One important question related to facial articulation is how much acceleration is possible. On our characters, which employ fairly complicated kinematic deformer networks, we have found that faces lie in between the near linear cost per point calculation and

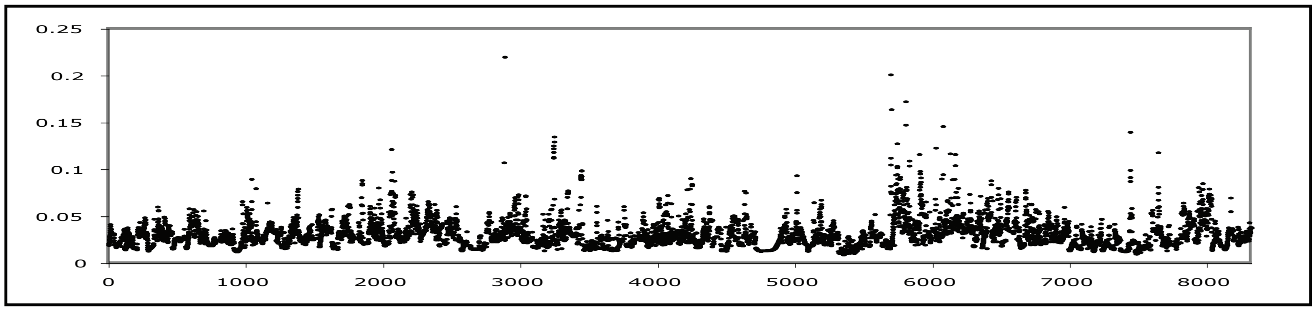

Figure 6: *The RMS error (in centimeters) using KPSA for each pose in the boy example. The error is less than 0.1 for all but a few poses.*

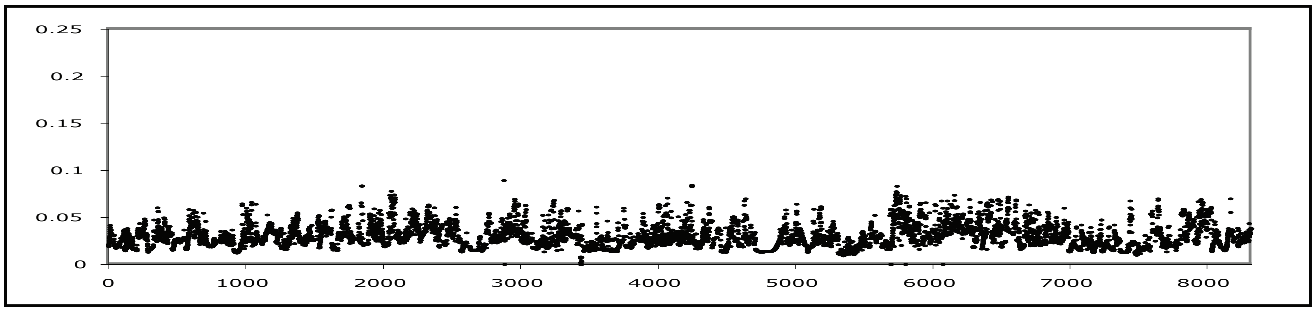

Figure 7: *The RMS Error (in centimeters) using the Soft Caching extension for each pose in the boy example. Notice how the error has been greatly reduced for the few poses that were problematic.*

a constant cost (where posing a small set of points is no less expensive than posing than the full face). For our boy character (Figure 1), we have found that the cost of posing our 170 key points is about 10% of the cost of posing the full 2986 points. In particular, all 2986 points can be posed in 0.5 seconds on average, while the 170 key points can be posed in 0.05 seconds on average. The KPSA approximation itself takes 0.00745 seconds on average resulting in an 8.7x speedup in posing time. We do not achieve a speedup linear with the key point to total point ratio for a few reasons: First, the key points often require more complicated deformations than the other points. For example, the mouth points require more effort to pose than the points on the back of the head. Second, multiple points can sometimes reuse computations - so posing twice the points won't necessarily require twice the computations. Once again we stress that this character was the most difficult and other characters that required fewer key points enjoy even more impressive speedups.

In order to validate our technique, we have examined the KPSA results using the independent test poses. Figure 1 shows an example of the quality of the KPSA result. The top-left image shows the character with all 2986 points posed using our in house posing engine, while the top-right shows the result of posing only the 170 key points with our in house posing engine and then using KPSA to generate the 2986 facial points. Notice that the KPSA computed result is an extremely accurate recreation of the fully computed pose. The accompanying video [Meyer and Anderson 2007] shows the side by side results for the more than the first 800 poses in the independent test set. As can be seen in the video, the KPSA computed results match the fully computed results very well - the maximum error for any point over 99.5% of the test poses is less than 1% of the diagonal of the head bounding box.

For most of the test poses the KPSA computed result and the fully computed result are visually similar and any differences are only noticeable through detailed examination of the side by side results. However, as noted in section 2.3, there are times when the estimated pose does not accurately match the fully computed pose. Figure 5 and the video show an example of one of the worst poses in our test set. The left image displays the pose generated by full computation, while the middle shows the result of the KPSA technique. Notice how the lips in particular do not match between the two techniques. In these cases, the Soft Caching mechanism (section 2.3) can detect and fix such poses. Using $\varepsilon = [0.1, 0.15]$, the Soft Cache detects the incorrect poses and generates a more accurate pose as can be seen on the rightmost image and in the video. With these settings, we have a cache hit rate of 96.6% for the entire dataset (8042 poses out of 8322 poses). Note that poses that cause a cache miss do pose slightly slower than simply computing a full pose (due to the subspace projection, error computation and possible interpolation), however it has been less than 5% slower in all our tests.

Figure 6 shows the RMS error (in centimeters) for each pose in the boy example. Notice how the error is less than 0.1 cm for all but a few of the poses. Figure 7 shows the RMS error using the Soft Caching extension. Notice how the error has been greatly reduced for the few poses that were problematic.

In addition to the boy example, we tested our facial articulation technique on a wide range of characters spanning 3 feature films, including additional human characters, a car character and a rat character. A selection of output poses from our KPSA technique are shown in Figure 8 for the car character and Figure 9 for the rat character, and statistics are given in Table 1. Since these characters did not have as much extreme and unpredictable motion as the boy, they required fewer basis vectors and key points and enjoyed larger speedups. These examples demonstrate the wide applicability of our technique.

Although our test characters all use a kinematic deformation rig, interesting effects can be achieved for characters using poses defined by physical simulation or hand sculpting. The user could define a version of the face that can be manipulated by the animator at runtime (the control face). Then we could create a training set containing pairs of the physically simulated (or hand sculpted) face and the corresponding pose of the control face. At runtime, the animator poses the control face and the system uses the control face points to find the projection into the joint (control face, simulated face) subspace and computes the corresponding simulated face. Note that in this "Cross KPSA" case the Soft Cache fail through to a full calculation is not possible.

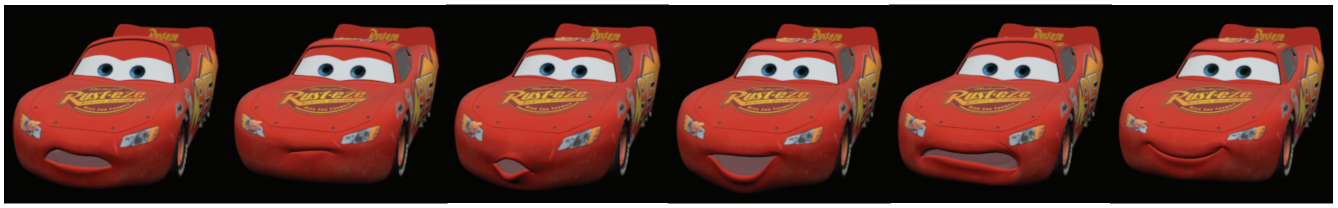
Figure 8: *A selection of output poses from our KPSA technique applied to a car's facial articulation (35 basis vectors, 70 key points). The KPSA (without Soft Caching) results were accurate with the max error below 5.3 mm and the rms error below 0.5 mm for the entire animation.*

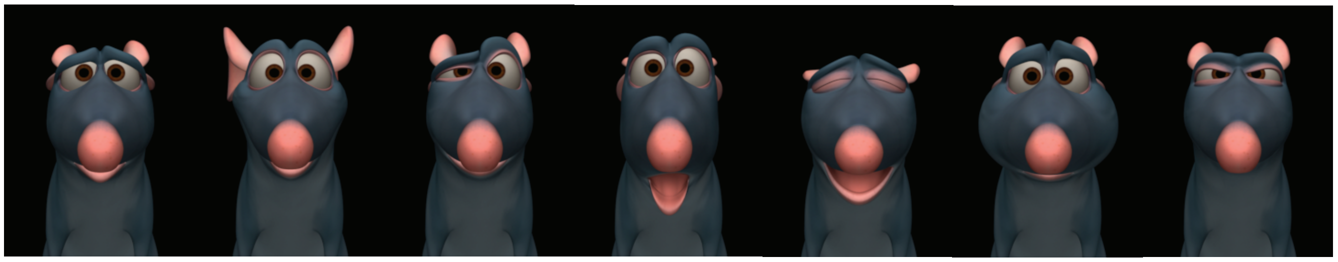
Figure 9: *A selection of output poses from our KPSA technique applied to a rat character's facial articulation (with fur removed for display). This model used 40 basis vectors and 80 key points. The KPSA (without Soft Caching) results were accurate with the max error below 1.0 mm and the rms error below 0.12 mm for the entire animation.*

3.2 Applications to Rendering

A second application for our acceleration technique is the computation of indirect illumination. This example is presented as a second application of the KSPA technique and is not meant to be a full-fledged rendering algorithm. Although there are many different acceleration schemes for indirect illumination, both using or not using subspaces (e.g. [Ward et al. 1988; Ward and Heckbert 1992; Sloan et al. 2003; Jensen 1996; Kristensen et al. 2005; Hasan et al. 2006; Meyer and Anderson 2006]), these types of rendering problems are potentially well suited to acceleration using KPSA since they often have a large fraction of the total computation in a final gather step whose computational costs vary nearly linearly with the number of points computed.

Using KPSA on indirect illumination works in much the same way as in facial articulation, except the values are now indirect illumination RGB triples at the points of the scene instead of position values. We constructed a training set by computing the indirect illumination resulting from placing a point light in 144 different locations in a room similar to the Cornell box (examples of the training images are shown in Figure 10(a)). Training results in 32 illumination basis vectors and 200 key points. At runtime, the indirect illumination is computed only at these 200 key points and KPSA is used to produce the final result. Figures 10(b,c,d) show comparisons of the fully computed illumination versus the KPSA computed illumination for a few select frames with novel lighting configurations - the KPSA computed illumination is visually very close to the full computation.

Note that the use of the lighting at key points as input to our system as opposed to lighting variables (such as light position) allows our system to handle changes in the lighting variables rather easily. For instance, changing the light type, or distance falloff, etc. will have a complex, nonlinear effect on the resulting indirect illumination. Since our system actually computes the indirect illumination at the key points and uses the illumination itself to drive the statistical model, we do not require a way to map each of these animation variables to the final indirect illumination as other methods would. In a production setting where lights have many complex and often interacting controls, this is a huge benefit. We are in the process of exploring the use of this sort of technique in interactive relighting including global illumination (analogous to the system described in [Hasan et al. 2006]).

4 Summary and Ongoing Work

In this paper we have presented KPSA - a method for accelerating certain calculations in computer graphics. Using KPSA, a computation is performed sparsely - only at automatically chosen key points - and a linear subspace is driven by these key point values to determine the values at the remaining points. By using the key point values to capture the nonlinearities of the space, the statistical model can compute the additional points without having to deal with these nonlinearities. The Soft Caching extension allows KPSA to be used like a cache, falling back to a full calculation in the event of a "cache miss".

The main limitation of our method is that, like many subspace methods, it is dependent on a good training set. If a runtime pose is required that is not adequately modeled by the training set, the KPSA approximation may not be accurate. Although Soft Caching reduces these types of problems, it does not alleviate the need for good training. Additionally, the acceleration is determined mostly by the time required to compute the values at the key points. For instance, if computing the values at a subset of points is no faster than computing the values at all points, KPSA is not applicable. One interesting area of future research is to examine alternate key point selection strategies, possibly trading off accuracy for acceleration.

Another area of ongoing work involves "local" fail through. In certain applications, it is reasonable to expect that subspace projection error should be local, confined to a particular unexercised animation control or local rendering feature such as a contact shadow. We are currently in the process of exploring the use of various forms of basis function rotation and clustering to be able to localize the projection error so that the fail through process would only need to compute the full calculation on a subset of the domain.

Acknowledgements: We would like to thank Disney, Pixar Animation Studios, Pixar Studio Tools, and Pixar Research for supporting this work.

References

GUENTER, B., GRIMM, C., WOOD, D., MALVAR, H., AND PIGHIN, F. 98. Making faces. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 55–66.

HARMON, H. H. 1976. *Modern factor analysis*, 3rd ed. University of Chicago Press.

	Bounding Box	Points	Poses	BasisVecs	KeyPoints	PCA Solve	Keypoint Solve	Speedup
Boy Head	24cm x 26cm x 28cm	2986	2503	85	170	349s	0.0075s	8.7x
Car	2.2m x 4.3m x 1.3m	2625	242	35	70	19.6s	0.0045s	15x
Rat	6.3cm x 28cm x 27cm	4150	245	40	80	51s	0.0100s	20.75x

Table 1: *Statistics using KPSA for facial articulation: Note that for our tests, all key points and basis vectors were automatically computed. All timings were performed on an Intel Xeon 3.4Ghz with 4GB of ram.*

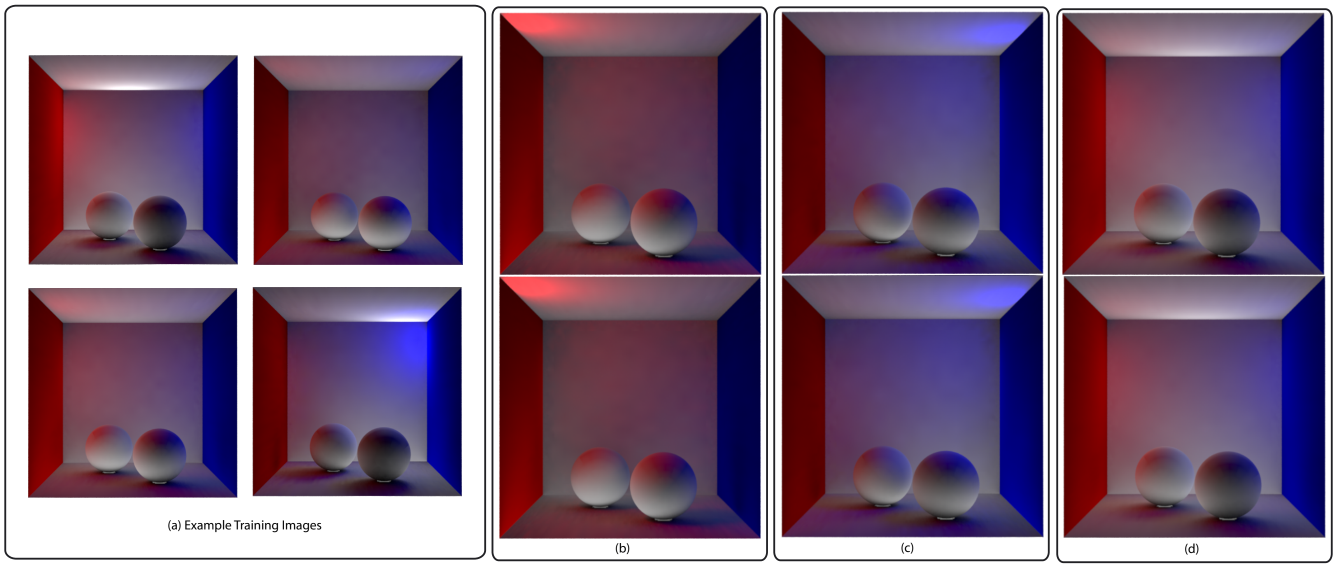

Figure 10: *An example of the KPSA technique applied to rendering indirect illumination: (a) Example images from the 144 training images. (b,c,d) Comparisons of the fully computed solution using 160000 points (top) to the KPSA solution computed from the 200 key point values (bottom) for novel light positions. Note that the KPSA results are visually similar.*

HASAN, M., PELLACINI, F., AND BALA, K. 2006. Direct-to-indirect transfer for cinematic relighting. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 1089–1097.

HWANG, B.-W., BLANZ, V., VETTER, T., AND LEE, S.-W. 2000. Face reconstruction using a small set of feature points. In *Biologically Motivated Computer Vision*, 308–315.

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

JAMES, D. L., AND PAI, D. K. 2002. Dyrt: dynamic response textures for real time deformation simulation with graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 582–585.

JENSEN, H. W. 1996. Global illumination using photon maps. In *Rendering Techniques '96 (Proceedings of the 7th Eurographics Workshop on Rendering)*.

KRISTENSEN, A. W., AKENINE-MOELLER, T., AND JENSEN, H. W. 2005. Precomputed local radiance transfer for real-time lighting design. *ACM Transactions on Graphics (SIGGRAPH 2005) 24*, 3, 1208–1215.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 153–159.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 165–172.

MEYER, M., AND ANDERSON, J. 2006. Statistical acceleration for animated global illumination. *ACM Transactions on Graphics (SIGGRAPH 2006) 25*, 3, 1075–1080.

MEYER, M., AND ANDERSON, J. 2007. Key point subspace acceleration and soft caching. Tech. Rep. 06-04b, Pixar Animation Studios. `http://graphics.pixar.com/SoftCachingB/`.

MO, Z., LEWIS, J., AND NEUMANN, U. 2004. Face inpainting with local linear representations. In *BMVC*.

SHASHUA, A., AND WOLF, L. 2004. Kernel feature selection with side data using a spectral approach. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3, 382–391.

WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 129–138.

WARD, G., AND HECKBERT, P. 1992. Irradiance Gradients. In *Proceedings of the 3rd Eurographics Workshop on Rendering*, 85–98.

WARD, G., RUBINSTEIN, F., AND CLEAR, R. 1988. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics (ACM SIGGRAPH '88 Proceedings)*, 85–92.

ZHANG, Q., LIU, Z., GUO, B., AND SHUM, H. 2003. Geometry-driven photorealistic facial expression synthesis. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 177–186.