Compression and Direct Manipulation of Complex Blendshape Models

Jaewoo Seo* KAIST
Geoffrey Irving† Weta Digital
J. P. Lewis‡ Weta Digital
Junyong Noh§ KAIST

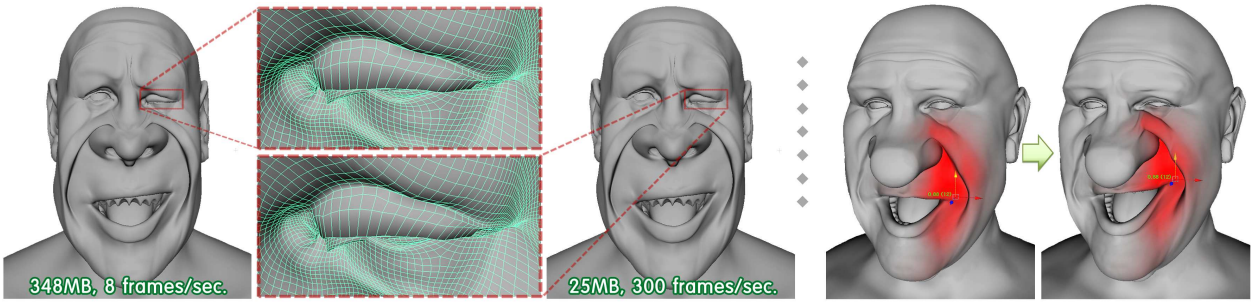


Figure 1: *Left: This facial model has 348 MB of uncompressed blendshape data, and runs at 8 frames per second on 8 CPUs. Our compression method reduces the storage to 25.4 MB and achieves 300 frames per second with a GPU implementation. No artifacts are visible. Right: The user can interactively manipulate the blendshape puppet by dragging any vertex on the model. The deforming region is colored red for visual feedback.*

Abstract

We present a method to compress complex blendshape models and thereby enable interactive, hardware-accelerated animation of these models. Facial blendshape models in production are typically large in terms of both the resolution of the model and the number of target shapes. They are represented by a single huge blendshape matrix, whose size presents a storage burden and prevents real-time processing. To address this problem, we present a new matrix compression scheme based on a hierarchically semi-separable (HSS) representation with matrix block reordering. The compressed data are also suitable for parallel processing. An efficient GPU implementation provides very fast feedback of the resulting animation. Compared with the original data, our technique leads to a huge improvement in both storage and processing efficiency without incurring any visual artifacts. As an application, we introduce an extended version of the direct manipulation method to control a large number of facial blendshapes efficiently and intuitively.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: blendshape, compression, direct manipulation

Links: DL PDF

*e-mail: goongsang@kaist.ac.kr
†e-mail: irving@naml.us
‡e-mail: zilla@computer.org
§e-mail: junyongnoh@kaist.ac.kr

1 Introduction

Blendshapes are a predominant technique for the creation of realistic and semi-realistic facial animation. The strength of the blendshape method lies in its simple and interpretable parameterization: the artist defines an 'expression space' with shapes such as raise-inner-brow-right, and then models each shape exactly as desired. Blendshape deformers are available in most commercial graphics packages such as Autodesk Maya and 3ds Max, and virtual characters created with this technique abound in recent popular films.

The blendshape formulation represents the face as a linear combination of a set of predefined shapes (i.e. morph targets),

$$o = \hat{B}w$$

where o is a vector containing the resulting vertex positions and w is a vector storing the blending weights. $\hat{B}$, the *blendshape matrix*, has the vertex positions of target shapes at its columns. In practice, the delta blendshape formulation is often utilized. In this form the targets B are offsets from a neutral shape vector n:

$$o = n + Bw \tag{1}$$

Typically, each target shape deforms only some part of the face, so this approach exposes sparsity in the matrix B. Our blendshape model is based on this formula.

One downside of the blendshape method is that it requires a large number of target shapes to produce high quality facial animation. For example, the *Gollum* character in *the Lord of the Rings* movies had over 900 target shapes [Raitt 2004]. Contemporary film-resolution models, such as the examples in this paper, generally require more than 1000 target shapes. With a high-resolution facial mesh, the result is a large blendshape matrix B with at least tens of thousands of rows and hundreds of columns, requiring hundreds of megabytes of memory. Although the processing involves only a matrix-vector multiplication, this size is too large to achieve real-time performance on current hardware. For example, after incorporating a blendshape model similar to that in Figure 1 into a realistic "rig" with additional deformers (e.g. for the neck), a coarse body model, textures, another character, and an environment, the resulting scene plays at less than 1 frame per second.

Storing the matrix in a sparse format is a natural choice. However, the matrices used in our examples have densities around 30%, so sparse storage saves little space as additional indices must be stored alongside the values. Principal component analysis (PCA) is another alternative. Unfortunately, the spectrum of the matrix decays too slowly to make PCA effective (Figure 2). For example, representing the matrix to relative error tolerance $\tau = 0.01$ gives a compression ratio of 34%. This is barely more than the initial sparse matrix, but is not accurate enough for high quality results.

The proposed method compresses these matrices much more efficiently than the above approaches. Our method is based on hierarchically semi-separable (HSS) representations [Chandrasekaran et al. 2004; Xia et al. 2010] that recursively decompose a block matrix into its diagonal elements and compressible low-rank off-diagonal blocks. While the original HSS is designed to work with square matrices with a dense-in-diagonal property, our blendshape matrices have rectangular shapes with irregular density patterns. To address this, we decompose the matrix into blocks and reorder them for compatibility with HSS, then apply our modified HSS compression (Figure 3). This leads to our new blendshape formula with two additional permutation matrices P and Q:

$$o = n + PHQw \quad (PHQ \approx B) \tag{2}$$

where H is the blendshape matrix compressed with HSS. P and Q are stored as permutation vectors, so their size is negligible compared to that of B or H. Applying the banded Householder transform [Irving 2011] to the rotation matrices in H leads to additional compression. With this combined scheme, we achieve a compression ratio of under 10% as shown in Section 5. Since a direct implementation of the matrix-vector multiplication in (1) is memory bound, this compression also improves performance. Moreover, the tree structure of HSS exposes natural parallelism, which allows efficient matrix-vector multiplication on either multi-core CPUs or GPUs. Note that the same method of matrix compression is applicable to any technique based on a large number of linear combinations. As an example, we apply our method to cage-based deformation [Ju et al. 2005; Joshi et al. 2007; Lipman et al. 2007] in Section 5.

The performance gain from matrix compression frees computation for other tasks, and we utilize this to provide an interactive tool that allows direct manipulation of the large number of blendshapes. As described in Section 4, many blendshape systems used in animation practice, including ours, partition the blendshape targets into a small set of primary targets and a larger set of secondary targets that are nonlinearly activated. Our direct manipulation approach accommodates such extensions and interoperates with the parameter editing provided in practical blendshape systems.

Although our research is focused on film applications that require the highest complexity blendshape models, games can also benefit from our technique. Current state-of-the-art games rely on linear blend skinning for facial animation [Ernst 2011], and models with even hundreds of targets are considered impossible due to low performance and high memory usage [Nguyen 2007]. Our method allows multiple characters of this complexity to fit in the graphics card memory and run in real-time, with computation to spare.

2 Related Work

Despite its long history of practical use [Bergeron and Lachapelle 1985], blendshapes were not generally recognized as a research area until the work of Pighin et al [1998]. The blendshape technique has been successfully integrated into various areas of facial animation, including modeling [Pighin et al. 1998], rigging [Li et al. 2010], facial motion capture [Joshi et al. 2003], and animation retargeting [Chuang 2004; Deng et al. 2006; Seol et al. 2011].

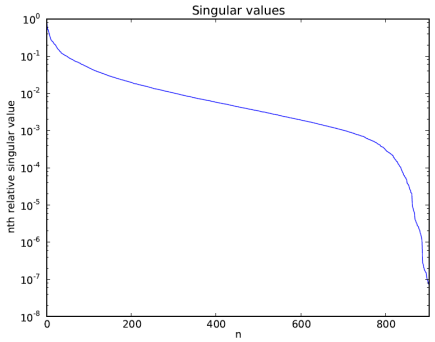


Figure 2: *The singular value spectrum of a blendshape matrix with $n \approx 1000$ target shapes. The singular value decays slowly until n reaches around 800, requiring a very large value of n for accurate PCA approximation.*

A variety of other techniques for facial deformation and animation exist in the research literature. Some important techniques include anatomically-based approaches [Sifakis et al. 2005] and sophisticated nonlinear deformers [Sumner et al. 2005; Feng et al. 2008]. In particular we wish to contrast data-driven approaches based on underlying PCA models [Blanz and Vetter 1999; Vlasic et al. 2005]. The PCA approach has clear advantages in terms of analysis and computation, but it sacrifices the high-level interpretable parameteriztion that is unique to blendshapes.

There is a broad range of research on mesh compression, for both static meshes [Karni and Gotsman 2000; Sorkine et al. 2005] and animation sequences [Stefanoski and Ostermann 2008]. A good discussion of this area can be found in [Peng et al. 2005]. As our solution is specialized for blendshapes, the proposed method differs from the existing work in two ways: (1) We are less concerned about compressing mesh connectivity, as the blendshape matrix contains vertex positions only. (2) Our method is also optimized for arbitrary interactive editing, while some existing methods focus on compressing predefined animation. Lastly, our application assumes "asymmetric" compression, in which significant computation can occur in the compression phase provided that the decompression is as fast (and memory local) as possible.

Direct manipulation has been considered as an effective interface for mesh deformation, as it provides simple and intuitive controls over complex meshes. Feng et al. [2008] proposed a deformation model based on a kernel canonical correlation analysis (kCCA) predictor. From a set of sparse example poses, users can generate plausible deformations by dragging the landmark vertices that are chosen in a training stage. Lau et al. [2009] designed a system to intuitively pose facial models. Along with the typical point constraints, they also introduced distance constraints and stroke constraints for more precise user control.

The problem of manipulating facial blendshapes is related to marker-driven facial capture [Zhang et al. 2004], in that manipulating a vertex corresponds to a marker constraint in motion capture. The difference is that the former is an under-constrained problem as only one vertex is manipulated at a time, whereas the latter deals with multiple markers per frame. Zhang et al. [2004] and Lewis and Anjyo [2010] addressed this issue, but neither approach applies to our system directly because of the nonlinearly activated targets mentioned above. Our solution subsumes the method of Lewis and Anjyo [2010] while handling the nonlinearity effectively.

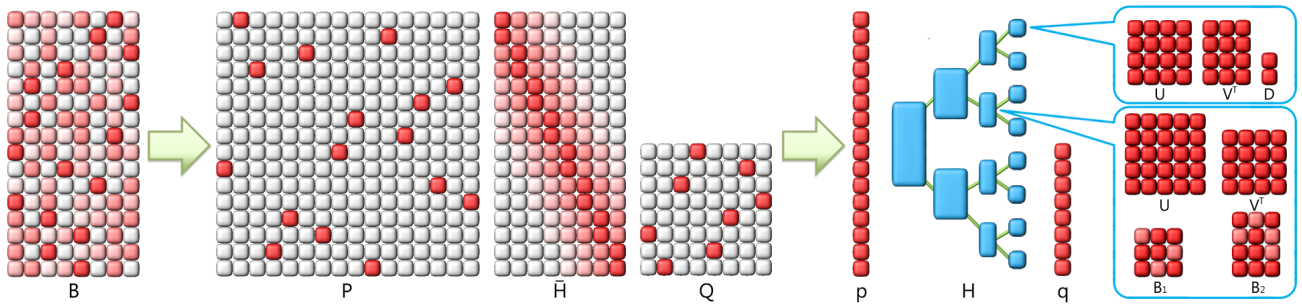


Figure 3: *Steps in the blendshape matrix compression. Dense matrix blocks are colored in red. The initial matrix B (left) is reordered for HSS compression, creating two permutation matrices, P and Q (center). The results are the HSS matrix H and the permutation vectors p and q (right). Each node of H has either three (leaf nodes) or four (non-leaf nodes) element matrices.*

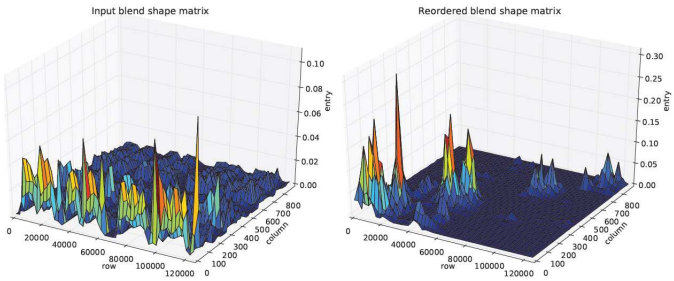
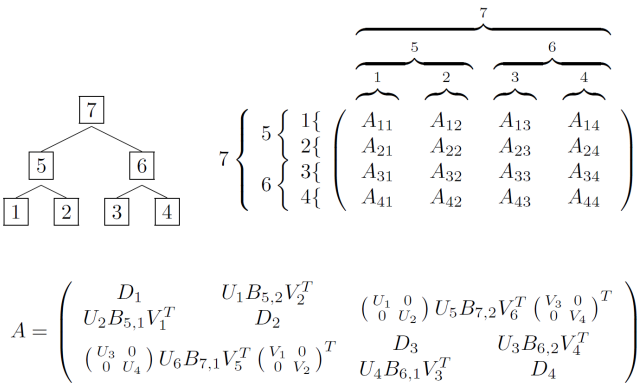


Figure 4: *Visualization of the blend shape matrix before (left) and after (right) reordering. Each entry in the plot is the average absolute value of a small matrix block.*



$$A = \begin{pmatrix} D_1 & U_1 B_{5,2} V_2^T & \begin{pmatrix} U_1 & 0 \\ 0 & U_2 \end{pmatrix} U_5 B_{7,2} V_6^T \begin{pmatrix} V_3 & 0 \\ 0 & V_4 \end{pmatrix}^T \\ U_2 B_{5,1} V_1^T & D_2 & \\ \begin{pmatrix} U_3 & 0 \\ 0 & U_4 \end{pmatrix} U_6 B_{7,1} V_5^T \begin{pmatrix} V_1 & 0 \\ 0 & V_2 \end{pmatrix}^T & & D_3 \quad U_3 B_{6,2} V_4^T \\ & U_4 B_{6,1} V_3^T & D_4 \end{pmatrix}$$

Figure 5: *Hierarchically semiseparable (HSS) representation corresponding to a 4×4 partition of a matrix*

3 Compression

The blendshape models we consider are both extremely large and resistant to compression using standard techniques such as PCA (Figure 2). However, there is still a great deal of smoothness and other structure in the set of blendshapes, as shown in Figure 4 (left). To take advantage of this structure, we apply the hierarchically semi-separable (HSS) representation of [Chandrasekaran et al. 2004], which compresses a matrix by hierarchically replacing off-diagonal blocks with low rank approximations. This technique is very attractive in that the low-rank criterion places few restrictions on the model and does not require a particular form of smoothness such as vanishing moments.

As the original HSS only deals with square matrices, we first reorder the rows and columns of the matrix to concentrate weights near the suitably defined "diagonal" for a rectangular matrix. The reordering and compression stages are described in details below.

3.1 Reordering

An example input matrix is shown in Figure 4 (left). Although there is a large amount of structure, it is not in the diagonal vs. off-diagonal form that is expected by the HSS algorithm. Indeed, the term *diagonal* requires definition as the matrix is not square. Leaving this definition loose for the moment, we want to reorder the rows and columns of the matrix to move low rank blocks away from the diagonal.

Unfortunately, any formulation in terms of the ranks of subblocks is likely to be NP hard and inapproximable [Çivril and Magdon-Ismail 2009]. Alternatively, if the matrix A is $m \times n$, we could define a weight function such as $w_{ij} = |n_i - m_j|$ that grows away from the diagonal and seek permutations p_i and q_j such that

$$E = \sum_{ij} w_{ij} |a_{p_i q_j}|$$

is minimized. However, this formulation is exactly the quadratic assignment problem (QAP), which is also NP hard and practically extremely difficult to solve [Loiola et al. 2007].

To find a more practical (but still NP hard) version of the reordering problem, we consider the matrix A as a 2×2 block matrix

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

As will be discussed in Section 3.2, each of the off-diagonal blocks A_{12}, A_{21} will be compressed into a single dense block by the HSS algorithm, so it is important that their ranks be small. We can make this practical by replacing rank by weight and seeking bisections $R = R_1 \sqcup R_2, C = C_1 \sqcup C_2$ of the rows and columns of A such that the crossing weight

$$E = \sum_{ij} |(A_{R_1,C_2})_{ij}| + |(A_{R_2,C_1})_{ij}|$$

is minimized. If the sizes of R_k and C_k are allowed to vary, this problem is equivalent to minimum cut, and admits efficient polynomial time solutions. With the partitions restricted to have equal size ($|R_1| = |R_2|$ and $|C_1| = |C_2|$), the problem is NP hard but admits a simple and efficient heuristic due to [Kernighan and

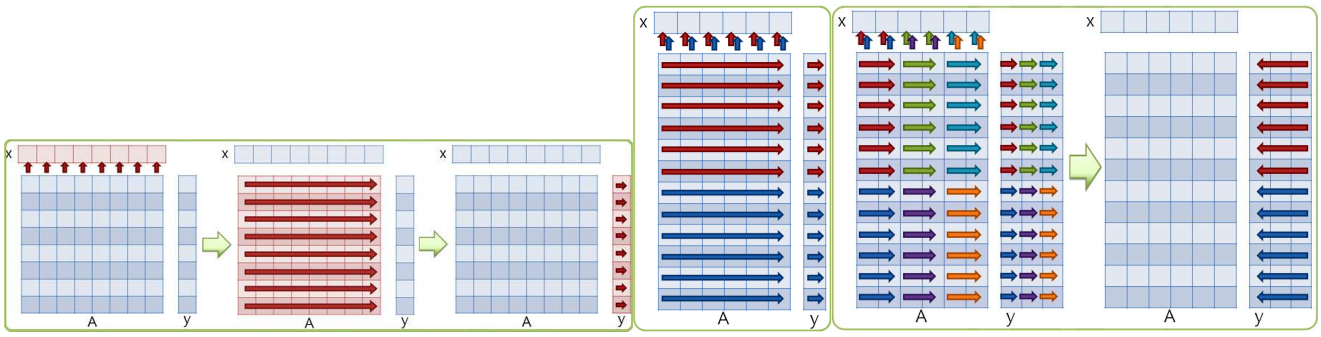


Figure 6: *(a) Matrix-vector multiplication ($y = Ax$) with a single thread block. The threads read x into shared memory (left), 'march' the matrix together row-wise (center), and write the result to y (right). The process can be divided by rows only (b) or by both rows and columns (c). Threads of the same color belong to the same thread block. In the row-column case, the second step (right) gathers scattered results from the first kernel (left) into the final result. The storage overhead of y is negligible as we divide the matrix into at most 4 blocks of columns.*

Lin 1970]. This heuristic iteratively improves the current bisection by first greedily computing a sequence of pairs to swap, and then choosing the optimal prefix of the greedy sequence to swap as a whole. Although the original paper discussed bisecting only a single set, the algorithm is easily generalizable to simultaneous bisection of rows and columns by lumping both into a single set and restricting the heuristic to only swap row-row or column-column pairs. We found that only a small constant number of iterations of this heuristic was required to produce a high quality bisection in all cases. Kernighan and Lin [1970] reported $O(n^2 \log n)$ time per iteration; we improve this to $O(n^2)$ time by replacing the sorting step in their algorithm with a lazy sort, for a total practical cost of $O(n^2)$ time per matrix bisection.

Once the initial row-column bisection is computed, we recurse into the two diagonal blocks A_{11} and A_{22}, applying the bisection heuristic to both. Each lower level involves twice as many bisections as the level above and each bisection is one quarter as expensive, so the time for the entire recursion is a geometric series summing to $O(n^2)$. This is expensive but not excessively so, as matrix reordering and compression are applied as a preprocess (Table 1).

Figure 4 (right) shows the result of applying the resulting permutations to our example input matrix. We restrict the permutations of the rows of the matrix to treat each vertex as a block, by applying the reordering step to the $m/3 \times n$ matrix given by taking the norm of each 3×1 block.

3.2 Hierarchically semiseparable matrices

Once the matrix has been reordered, we compress it using the HSS representation of [Chandrasekaran et al. 2004]. HSS is closely related to H-matrices [Börm et al. 2003], H^2-matrices [Börm 2010], hierarchical tensor approximations [Wu et al. 2008], and (more distantly) to fast multipole methods. It makes few assumptions on the underlying problem and provides several fast operations including matrix multiplication. We present a brief description here, and refer to [Chandrasekaran et al. 2004; Xia et al. 2010] for details on constructing and applying HSS representations.

To define an HSS representation, we first partition the rows and columns of the matrix into an equal number of blocks, and form a binary tree with the partitions as leaves and the entire matrix as the root. Although our matrices are nonsquare, partitioning into a square array of blocks allows us to apply the algorithms of [Chandrasekaran et al. 2004] as if they were square. Each leaf node k stores the diagonal block D_k from the original matrix together with rotations U_k on the left (resp. V_k on the right) that compress everything in that row (resp. column) *except* for the diagonal block. Further up the tree, each non-leaf k stores two matrices $B_{k,1}$ and $B_{k,2}$ that store everything that was not stored in its children after compression by the children's U's and V's, followed by its own U_k and V_k. Figure 5 shows an example HSS representation. Our notation differs slightly from [Chandrasekaran et al. 2004]: we stack each pair of their R (resp. W) matrices into a single U (resp. V) matrix, which reduces the number of function calls when computing matrix-vector products. When compressing each matrix block, we perform a singular value decomposition and drop all singular values satisfying $\sigma_i \leq \tau\sigma_1$, where τ is a relative error tolerance and σ_1 is the largest singular value of the block.

For additional compression, we optionally represent rotations using the banded Householder factorization of [Irving 2011]. Given an orthogonal $m \times n$ matrix U with $m \leq n$, the banded Householder factorization finds an orthogonal matrix U' with the same range that can be stored in $n(m-n)$ floats. By representing all U and V matrices in this form, the total cost of storing an HSS representation decreased by almost half (Table 1). On the other hand, this representation presents a memory-versus-speed tradeoff (see Section 5), so it is not used when speed is paramount.

3.3 Parallel Processing

HSS offers a fast multiplication algorithm exploiting its binary tree structure. It consists of two stages—an up sweep and a down sweep—that repeatedly update two auxiliary vectors f and g with a series of matrix-vector multiplications performed by the tree nodes at each level. The size of the matrix varies depending on the nodes with a tendency to get bigger as the level goes up: it can be as small as 1×1 at leaf nodes and as big as $thousands \times hundreds$ at the root. Consequently, there are different computation patterns at each level, with many small sized multiplications at lower levels and a few large sized multiplications at upper levels.

As updates in the same level are independent of each other, per-node parallelization can be easily achievable by assigning a thread to each equally-divided node group. For CPU-based implementation, this coarse parallelization scheme leads to near linear speedup. However, parallelization on GPUs requires a more elaborate approach for better utilization of the massively parallel processing architecture.

We use CUDA [NVIDIA 2010] for the GPU implementation. It offers two levels of hierarchy for parallel execution: threads and thread blocks. Individual threads form a thread block, a group of which, in turn, form a thread block grid. Threads in a same block can cooperate together through thread synchronization and data sharing with local memory. Although cooperation is not possible across the thread blocks, it is advantageous to have more blocks as this may allow the GPU to schedule the thread execution order more optimally. Keeping this in mind, we have designed various kernels, based on the type (dense or banded Householder) and size of the matrix. An appropriate kernel is chosen to achieve the optimal performance depending on the situation as described below.

Figure 6 illustrates the multiplication process between a dense block matrix and a vector. As this is a memory-bound operation, it is important to reduce the number of global memory operations and to hide memory latency by using many thread blocks. To do this, we keep the thread block size as small as possible and share the operand vector between threads in the same block. For matrices with more rows than the number of threads per block, multiple thread blocks are assigned so that each block processes a subset of the rows of the matrix. As the matrices typically have rectangular shapes with $m > n$, this approach is efficient for most cases. When the number of columns becomes large (bigger than the number of threads per block), we divide column-wise as well. Here additional treatment is necessary as the destination vector becomes a critical section. We use two different approaches, depending on whether the operand vector and the destination vector are the same. If they are different, the atomic add operation updates the destination vector. If they are the same, the former may cause read-after-write hazard, so we employ a two-step approach where the overlapping blocks write to different locations in the first kernel and a second kernel sums them into the final result. When the number of columns is large enough, the overhead of these extra operations is negligible [Bainville 2010].

For the matrices with banded Householder representation, it is difficult to divide a single matrix-vector multiplication into multiple blocks because each step depends on the previous one and results must be shared among the threads. It also requires more instructions and thread synchronizations than is required by dense matrix multiplication, making the operation computation bound. We reduced the instruction count and avoided dynamic branching as much as possible by unrolling loops and using compile-time optimizations.

4 Direct Manipulation

4.1 Nonlinear Blendshape Activation

Modern high-quality blendshape systems, including ours, partition the blendshape model into a smaller set of ≈ 100 *primary* targets that are controlled by the artist, and a remaining set of ≥ 1000 secondary targets that are automatically triggered as simple nonlinear functions of the movement of the primary shapes. This is done for several reasons: (1) to hide the complexity of the blendshape system from the animators, allowing them to focus only on manipulation of clearly defined primary parameters; (2) to add simple nonlinear behavior while presenting the artist with an interface in terms of traditional linear parameters. Although there is no strict guideline to define such functions, several *de facto* approaches exist [Osipa 2010; Autodesk 2011]. Types of these secondary shapes include:

- **Intermediate shapes** are useful when the primary shapes go through nonlinear deformation during the morphing. For instance, closing an eyelid exhibits arc-shaped movement on top of the underlying eyeball. In this case, an intermediate shape close_eyelid_left_50 is blended together with the primary shape close_eyelid_left when its weight is near 0.5, resulting in better in-between animation. Multiple intermediate shapes can be employed for a primary shape if desired.
- **Correction shapes** comprise the majority of the blendshapes. They are required to correct the blendshape interference problem [Lewis et al. 2005]. Each correction shape fixes an artifact caused by a certain combination of the shapes. The weight of the correction shape is determined by the participating shapes, e.g. the product of all the weights in our case.

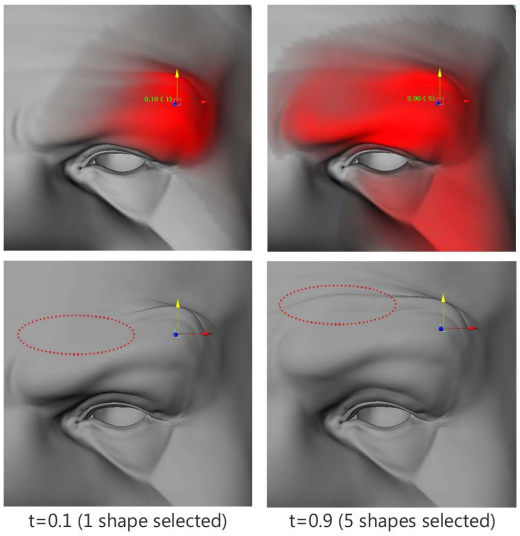


Figure 7: *The control of the influence region with different values of t. The regions are colored in red (top), leading to different manipulation results (bottom).*

Another important area to consider is the jaw. The jaw is deformed by both the blendshapes and the jaw bones with linear blend skinning (LBS). Like the secondary shapes, the same expression parameters that control the corresponding jaw blendshapes also drive the LBS joint transformations. In direct manipulation, we divide our total of 104 expression parameters into two groups, those that move the jaw bones (3) and those that do not (101). The two groups are solved independently. This is consistent with the practice adopted by the animators, as they usually work with the jaw bone first, followed by adjusting the other expression parameters.

4.2 Formulation

We start with an objective function similar to Lewis and Anjyo [2010]:

$$E = \frac{1}{2} \min_x \|\bar{B} f(x) - m\|^2 + \alpha \|x - x_0\|^2 + \mu \|x\|^2$$

Here x_0 and x are the expression parameters before and after the current manipulation, m is a vector of goal positions of the constrained vertices, $\bar{B}$ are the corresponding rows from the B matrix, and $w = f(x)$ is our nonlinear weight function. The first term measures the position differences between constrained vertices and their goals, while the second and third terms are for regularization. The last term with small μ avoids extreme solutions in the result.

Users often drag the manipulation vertex (referred to as a *handle* below) back and forth until the facial expression converges to a desired state. In terms of the mouse events, this can be described as a sequence like 'click-drag-drag-...-drag-release'. When the handle

is near the clicked position, the expression should be close to the initial state even after several drags. However, each drag should result in a continuous change relative to the previous state to ensure smooth changes in expressions. Setting x_0 on either the click or drag event will not simultaneously satisfy both conditions: if x_0 is set on click, the first condition is satisfied but the expression may change erratically during dragging. On the other hand, setting x_0 on drag will show smooth pose updates, but the facial state may drift away from the original pose after several drags. For this reason, we add a new regularization term x_d to our objective function:

$$\frac{1}{2}\min_x \|\bar{B}f(x) - m\|^2 + \alpha\|x - x_0\|^2 + \beta\|x - x_d\|^2 + \mu\|x\|^2$$

x_0 is set on the click event while x_d is set on every drag event.

After taking derivatives, this can be efficiently solved with any constrained nonlinear optimization solver. In our experience, the Bound Optimization BY Quadratic Approximation (BOBYQA) algorithm [Powell 2009] converged well with the fewest iterations. We used $\alpha = 0.3$, $\beta = 0.6$, and $\mu = 0.001$ in our examples.

We map the group of three jaw-related parameters to the position of a vertex in the jaw region using biharmonic radial basis function (RBF) interpolation [Carr et al. 2001]. This precomputes the RBF weights w with the positions of a handle vertex as input and the jaw parameters as output. At runtime, the handle is fixed on the trained vertex and the new jaw weights are predicted through this RBF network.

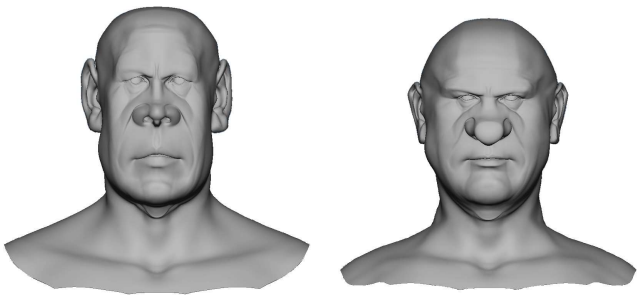
4.3 Region of Influence Control

Since the artist manipulates only a single handle at a time, direct manipulation of the face is highly underconstrained. For instance, when a user places the handle in outer brow region and moves the handle up, the user may or may not want to move the nearby inner brow as well. One solution would be to regularize this inverse problem by incorporating a prior based on the expected statistics of face movement [Lau et al. 2009]. (We note that the regularization terms in our formulation are in fact an approximate prior of this form, since the blendshape model is carefully constructed so that important expressions can be obtained with economical parameter changes.) In this work we adopt a solution that is suited for professional animators, allowing them to directly select a desired subset of blendshapes. The blendshapes that affect the handle are sorted by importance, and the user interactively selects a proportion $t \in [0, 1]$ of these shapes to manipulate. This is done in the interface by dragging the mouse with holding the 'control' key.

We define the importance of a blendshape relative to a vertex as the Euclidean norm of the vertex displacement vector from each shape. When a new manipulation vertex is set, the importance vector c can be efficiently computed from the corresponding rows of B. Let c_{max} denotes the maximum value of c. The subset S of blendshapes for a given t is defined as

$$s_i \in S \iff c_i \geq c_{max}(1 - t)$$

where s_i is the primary shape corresponding to c_i. Only expression parameters affecting the shapes in S take part in the optimization. Our definition of t provides users with an intuitive feeling when switching between local and global manipulation: a small t decreases the number of active shapes with a reduced deformation area; a large t widens it. For better user interaction, we visualize the influence region with a custom color shader (Figure 7).



(a) *Dumb (42391 verts)* **(b)** *Dumber (51729 verts)*

Figure 8: *Example facial models in the rest pose*

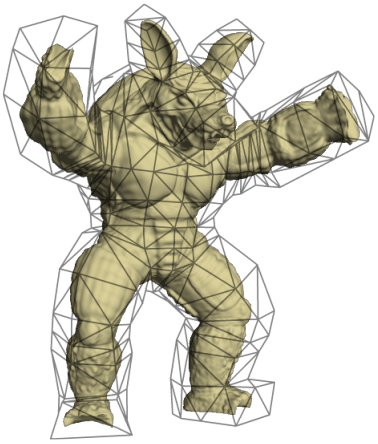


Figure 9: *An example model (Armadillo) for cage-based deformation in the rest pose (284 cage verts and 106289 embedded verts)*

5 Results

We implemented our method as a plugin for Maya 2011 with C++, Python, and *Maya Embedded Language* (MEL). A standalone facial animation player was also built to better demonstrate the performance improvement, as the Maya environment adds significant computational overhead. OpenMP and CUDA were used for parallelization of the matrix-vector multiplication and direct manipulation. The Intel Math Kernel Library (MKL) was used for all CPU-side BLAS and LAPACK operations. The BOBYQA implementation in the NLopt library [Johnson 2010] was used for nonlinear optimization of the direct manipulation. All the tests were performed on 8-core Intel Xeon 2.8Ghz machine with 8 GB of RAM and an nVidia GTX 580 GPU. With the exception of the direct manipulation solver, which uses double precision, all computation was done using single precision floats. Figure 8 and 9 shows the test models. For facial examples, we only utilized the main facial part of the mesh; the eyes and the teeth are not included.

Compression Table 3 shows compression results for different tolerances. A tolerance of $\tau = 10^{-3}$ resulted in no visible artifacts (Figure 1, 10 and 11), and was used for all examples. We performed a blind test with 9 professional artists who found no major differences between compressed and uncompressed models; four thought the compressed model was the ground truth. Table 2 shows storage and time requirements for the matrix compression stage. All the examples show a compression ratio of under 10% relative to the original matrix. Reordering took the most of the time in the compression stage, followed by HSS and banded Householder compression.

Example	Matrix Size # Rows	# Cols	Storage (MB) Dense	Sparse	PCA	local-PCA	HSS	HSS+banded
Dumb	127173	730	354 (100%)	348 (98.3%)	138.7 (39.2%)	87.4 (24.7%)	46.8 (13.2%)	25.4 (7.2%)
Dumber	155187	625	370.0 (100%)	317 (85.7%)	164.6 (44.5%)	104.7 (28.3%)	46.0 (12.4%)	28.1 (7.6%)
Armadillo	106289	284	115.2 (100%)	173.2 (150.3%)	114.8 (99.6%)	–	10.4 (9.0%)	8.6 (7.4%)

Table 1: *Storage information of the examples. $\tau = 10^{-2}$ is used for PCA and $\tau = 10^{-3}$ is used for our technique. Localized PCA with four pre-partitioned regions—eyes, nose, mouth and remainder—is also tested with the facial examples.*

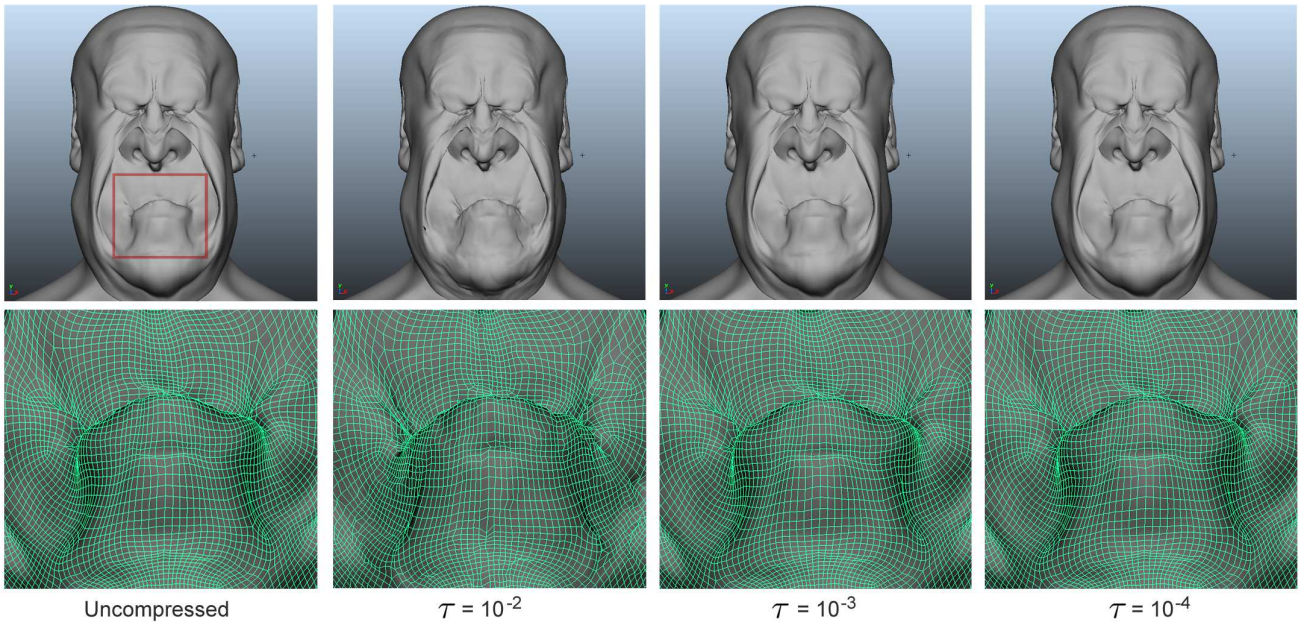


Figure 10: *The comparison of visual artifacts with varying tolerance τ. All expression parameters are set to 1 to test extreme deformation.*

Example	Reorder	HSS	banded
Dumb	670.2	133.2	3.4
Dumber	1031.3	142.4	3.5
Armadillo	6318.1	170.1	2.0

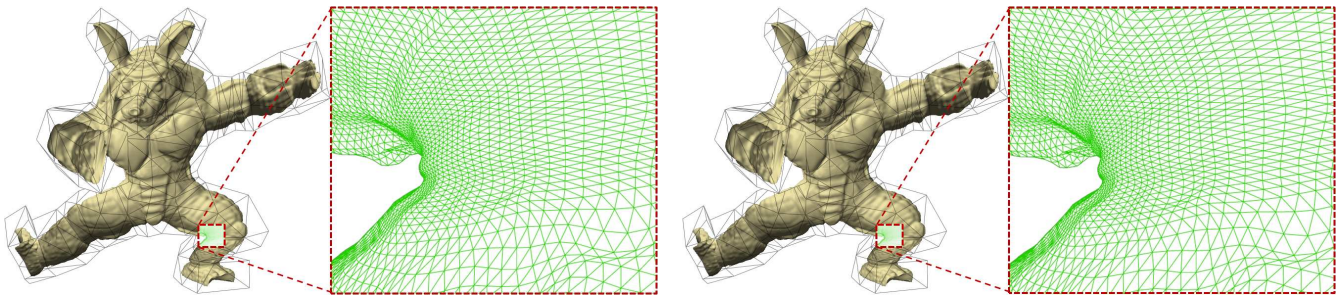
Table 2: *Detailed timing of our compression method (in seconds)*

Tables 1, 4 and Figure 11 show the fidelity and performance obtained with the compressed models. Unlike PCA, our method does not depend on the rank of the matrix. As a result, we show better compression ratios with lower relative error than PCA and localized PCA for all examples. To show the versatility of our technique, we also applied the proposed compression to cage-based deformation, which relies on a large number of linear combinations. Although, the matrix for *Armadillo* model is nearly incompressible with PCA, our results were comparable to the blendshape examples in terms of compression ratio, visual quality, and performance. Unlike the case of blendshapes, the cage deformation matrix has the same number of rows as vertices, so reordering is performed on the individual matrix elements and takes somewhat longer.

During matrix-vector multiplication, the number of flops is slightly less than twice the number of matrix entries. Consequently, the compression ratio approximates the amount of speedup (Table 3 and 4). GPU timing in Table 4 show that the banded Householder form caused a performance penalty (about 50%) due to the increased number of instructions and thread synchronizations, as described in Section 3. However, our standalone facial animation demo shows that it is already efficient enough for real-time applications: when fully processed on the GPU, our facial animation example runs at over 300 fps, including linear blend skinning and per-vertex normal updates (see Figure 1). Further speedup can be achieved by trading off between storage and performance if desired, by converting only a subset of the HSS tree to banded Householder format.

Direct Manipulation Figure 13 and the video show the advantage of using the dual regularizers in the objective function. The x_0 regularizer prevents the solution from drifting away from the initial state set on mouse click, while the x_d regularizer keeps the intermediate solutions smooth and continuous during mouse dragging.

Our direct manipulation method can reduce both the number of mouse actions and 'trial-and-error' iterations relative to traditional blendshape parameter editing. This is particularly true for novice users given the daunting number (> 100) of parameters. The accompanying video and Figure 12 show an example. On the other hand, it should be noted that in professional animation, direct manipulation cannot fully replace parameter editing. It can be mathematically argued that direct manipulation and parameter editing are each more efficient for some edits and inefficient for others [Lewis and Anjyo 2010]. An advantage of our approach is that it interoperates with conventional parameter editing, unlike approaches based on an underlying PCA model that lacks semantic interpretation.



(a) *Uncompressed* **(b)** *Compressed* ($\tau = 10^{-3}$)

Figure 11: *Quality comparison: cage-based deformation*

τ	Storage (Compression Ratio)	Rel. Error	Timing (ms) 8 CPUs	GPU
10^{-2}	9.83 MB (2.8%)	4.10%	5.58	1.12
10^{-3}	25.4 MB (7.2%)	0.54%	11.22	2.18
10^{-4}	54.3 MB (15.3%)	0.06%	19.3	3.39

Table 3: *Compression ratios, relative errors, and matrix-vector multiplication timing for various compression levels τ on the Dumb character*

Character	Sparse 8 CPUs	HSS 8 CPUs	HSS GPU	HSS+banded 8 CPUs	HSS+banded GPU
Dumb	124.21	11.22	1.47	10.73	2.18
Dumber	113.65	11.01	1.41	10.76	2.12
Armadillo	28.78	3.62	0.82	4.22	1.18

Table 4: *Timing for matrix-vector multiplication (in milliseconds)*

Feedback from professional animators was positive. They mostly liked the intuitive aspect of the method, which resembles that of inverse kinematics. Some commented on the control of the influence region; they could quickly grasp the level of control provided and were impressed by how our method "automatically" chooses the blendshapes that correspond to a desired direct manipulation edit. They mentioned that the order coincided with their expectation in most cases. Negative comments were mostly about the user interface. For example, many artists wanted to see a textual list of the shapes currently being manipulated, for verification. Some artists wanted a symmetry option, allowing manipulation of both the left and the right side of the face at the same time. Fortunately, adding such features will not adversely affect our core implementation.

Limitation Our method works well with complex models. However, the effectiveness of the compression depends on the size of the matrix. The HSS technique expects a certain level of smoothness in the input, which is often not observed in matrices with few rows and columns. We observed this by compressing a subset of the original blendshapes for the *Dumb* character, giving a compression ratio of 14.8% with 300 shapes and 47% with 50 shapes.

6 Conclusions and Future Work

We presented a method to compress complex blendshapes, using matrix block reordering followed by HSS and optional banded Householder representation. Animation with the compressed matrix is accelerated using fast matrix-vector multiplication and parallel processing. The resulting performance gain enables animation at interactive rates. We exploit this, introducing an interactive direct manipulation approach that handles our nonlinear blendshape scheme. Our direct manipulation method satisfies both novice and professional users due to its intuitive, efficient, and precise control and interoperability with traditional blendshape parameter editing. Applications include both professional off-line facial animation and real-time games with cinematic quality facial animation.

Our future direction targets improvement of both quality and performance. We would like to design a preconditioner that effectively reduces the block boundary artifacts from HSS compression. This might allow a higher error tolerance (e.g. $\tau \geq 10^{-2}$) to be used without visible artifacts. Another goal is to find a more GPU-friendly multiplication procedure for banded Householder matrices. Although the current method is already fast enough for our purpose, further improvement will allow computation of the facial movement of small crowds of characters in real time.

Acknowledgements

We thank Nebojsa Dragosavac for showing that a GPU-based facial puppet has merit. We also thank Jacob Luamanuvae, Marco Barbati, Ron Miller, Roger Wong, Sebastian Sylwan, and Joe Letteri for help and support in bringing this work into production use, and Yeongho Seol, Hyungjin Kim, Yewon Kim and Jungjin Lee for creating the examples in this paper. This work was partially supported by KOCCA/MCST (2-10-7602-003-10743-01-007).

References

AUTODESK, 2011. Autodesk Maya API White Paper.

BAINVILLE, E., 2010. OpenCL Training Course: GPU matrix-vector product . http://www.bealto.com/.

BERGERON, P., AND LACHAPELLE, P. 1985. Controlling facial expressions and body movements in the computer generated animated short 'Tony de Peltrie'. In *SIGGRAPH 85 Tutorial Notes, Advanced Computer Animation Course*.

BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3d faces. In *Proc. of SIGGRAPH 99*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 187–194.

BÖRM, S., GRASEDYCK, L., AND HACKBUSCH, W. 2003. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements 27*, 5, 405–422.

BÖRM, S. 2010. *Efficient numerical methods for non-local operators: H2-matrix compression, algorithms and analysis*. EMS Tracts in Math. European Mathematical Society.

CARR, J. C., BEATSON, R. K., CHERRIE, J. B., MITCHELL, T. J., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. In *Proc. of SIGGRAPH 2001*, ACM, Computer Graphics Proceedings, Annual Conference Series, 67–76.

ÇIVRIL, A., AND MAGDON-ISMAIL, M. 2009. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theor. Comput. Sci. 410* (November), 4801–4811.

CHANDRASEKARAN, S., GU, M., AND PALS, T. 2004. Fast and stable algorithms for hierarchically semi-separable representations. Tech. rep., University of California, Berkeley.

CHUANG, E. S. 2004. *Analysis, synthesis, and retargeting of facial expressions*. PhD thesis, Stanford, CA, USA. AAI3128633.

DENG, Z., CHIANG, P.-Y., FOX, P., AND NEUMANN, U. 2006. Animating blendshape faces by cross-mapping motion capture data. In *Proc. of the 2006 Symp. on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '06, 43–48.

ERNST, J., 2011. Fast and efficient facial rigging. Talk at Game Developers Conference (GDC) 2011.

FENG, W.-W., KIM, B.-U., AND YU, Y. 2008. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Trans. Graph. 27* (August), 91:1–91:9.

IRVING, G. 2011. Banded householder representation of linear subspaces. `http://arxiv.org/abs/1108.5822`.

JOHNSON, S. G., 2010. The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt.

JOSHI, P., TIEN, W. C., DESBRUN, M., AND PIGHIN, F. 2003. Learning controls for blend shape based realistic facial animation. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comp. Anim.*, Eurographics Association, SCA '03, 187–192.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26* (July).

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24* (July), 561–566.

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *Proc. of SIGGRAPH 2000*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 279–286.

KERNIGHAN, B. W., AND LIN, S. 1970. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Tech. J. 49*, 1, 291–307.

LAU, M., CHAI, J., XU, Y.-Q., AND SHUM, H.-Y. 2009. Face poser: Interactive modeling of 3d facial expressions using facial priors. *ACM Trans. Graph. 29* (December), 3:1–3:17.

LEWIS, J., AND ANJYO, K.-I. 2010. Direct manipulation blendshapes. *IEEE Comp. Graph. and Appl. 30*, 4 (July-Aug.), 42 –50.

LEWIS, J. P., MOOSER, J., DENG, Z., AND NEUMANN, U. 2005. Reducing blendshape interference by selected motion attenuation. In *Proc. of the 2005 Symp. on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '05, 25–29.

LI, H., WEISE, T., AND PAULY, M. 2010. Example-based facial rigging. *ACM Trans. Graph. 29* (July), 32:1–32:6.

LIPMAN, Y., KOPF, J., COHEN-OR, D., AND LEVIN, D. 2007. Gpu-assisted positive mean value coordinates for mesh deformations. In *Proc. of the fifth Eurographics Symp. on Geom. Proc.*

LOIOLA, E. M., MARIA, N., ABREU, M., BOAVENTURA-NETTO, P. O., HAHN, P., AND QUERIDO, T. 2007. An analytical survey for the quadratic assignment problem. *European Journal Operational Research 176*, 657–690.

NGUYEN, H. 2007. *GPU Gems 3*. Addison-Wesley Professional.

NVIDIA. 2010. *CUDA Compute Unified Device Architecture - Programming Guide*.

OSIPA, J. 2010. *Stop Staring: Facial Modeling and Animation Done Right, 3rd Ed.* Sybex.

PENG, J., KIM, C.-S., AND KUO, C.-C. J. 2005. Technologies for 3d mesh compression: A survey. *Journal of Visual Communication and Image Representation 16*, 6, 688 – 733.

PIGHIN, F., HECKER, J., LISCHINSKI, D., SZELISKI, R., AND SALESIN, D. H. 1998. Synthesizing realistic facial expressions from photographs. In *Proc. of SIGGRAPH 98*, ACM, New York, NY, USA, 75–84.

POWELL, M. J. D. 2009. The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. rep., Cambridge, England.

RAITT, B., 2004. The making of gollum. Presentation at U. Southern California Institute for Creative Technologies' *Frontiers of Facial Animation* Workshop, August.

SEOL, Y., SEO, J., KIM, P. H., LEWIS, J. P., AND NOH, J. 2011. Artist friendly facial animation retargeting. *ACM Trans. Graph. (Proceedings of SIGGRAPH ASIA 2011) 30*, 6.

SIFAKIS, E., NEVEROV, I., AND FEDKIW, R. 2005. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph. 24* (July), 417–425.

SORKINE, O., COHEN-OR, D., IRONY, D., AND TOLEDO, S. 2005. Geometry-aware bases for shape approximation. *IEEE Trans. on Viz. and Comp. Graph. 11*, 2 (March-April), 171 –180.

STEFANOSKI, N., AND OSTERMANN, J. 2008. Spatially and temporally scalable compression of animated 3d meshes with MPEG-4/FAMC. In *ICIP '08 - IEEE International Conference on Image Processing*, vol. 0.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24* (July), 488–495.

VLASIC, D., BRAND, M., PFISTER, H., AND POPOVIĆ, J. 2005. Face transfer with multilinear models. *ACM Trans. Graph. 24* (July), 426–433.

WU, Q., XIA, T., CHEN, C., LIN, H.-Y. S., WANG, H., AND YU, Y. 2008. Hierarchical tensor approximation of multidimensional visual data. *IEEE Trans. on Vis. and Comp. Graph. 14* (January), 186–199.

XIA, J., CHANDRASEKARAN, S., GU, M., AND LI, X. S. 2010. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications 17*, 6, 953–976.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2004. Spacetime faces: high resolution capture for modeling and animation. *ACM Trans. Graph. 23* (August), 548–558.

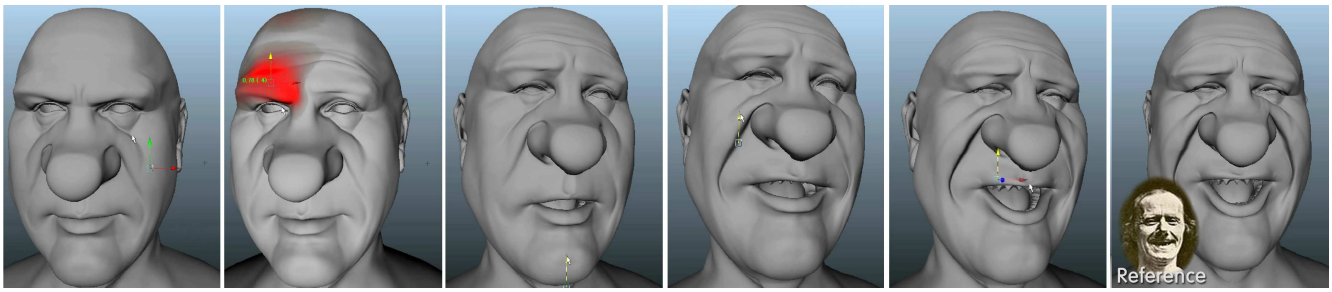


Figure 12: *An example of creating a facial expression by a novice user using our direct manipulation method. It took less than 3 minutes to create this from the neutral pose.*

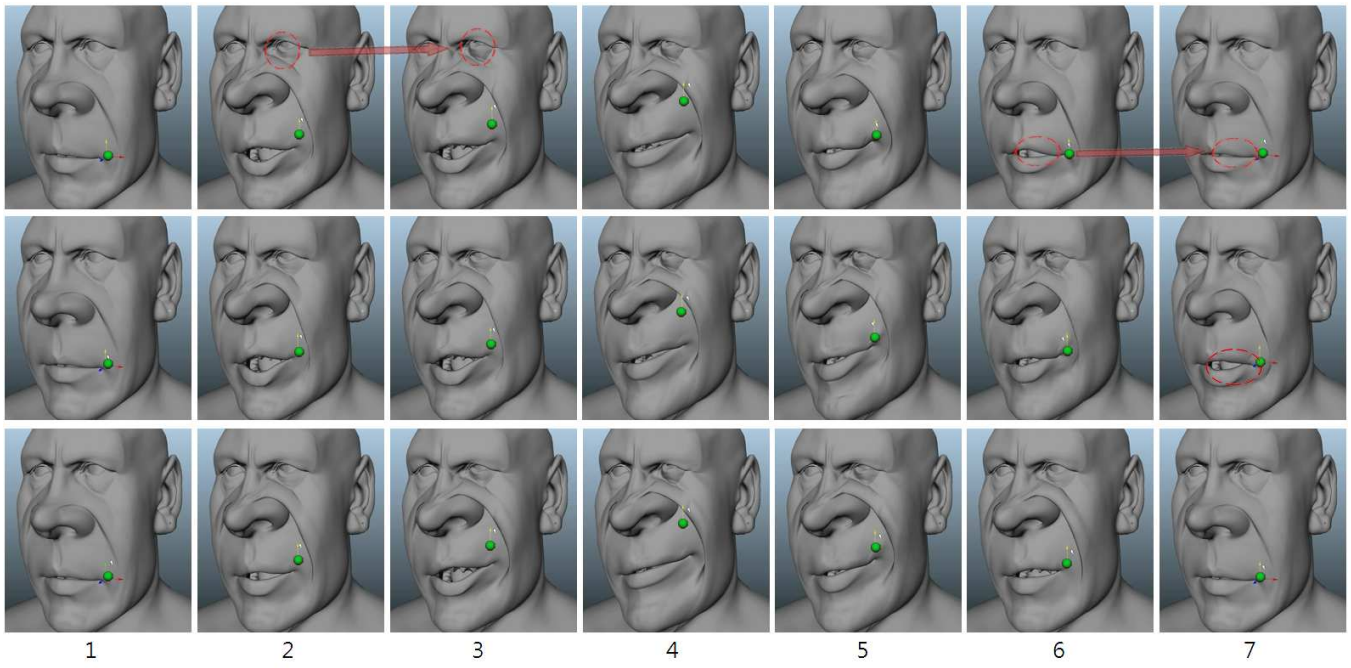


Figure 13: *Comparison between different regularizations used for direct manipulation. The handle positions are indicated with green spheres. The mouse is clicked on the first frame (first column), dragged up and down, and returned to the initial position (last column). Using only the x_0 regularizer fails to produce a continuous solution during the drags, causing wiggling movements and a sudden jump in the solution (top row, column 2-3 and 6-7). Using only the x_d regularizer shows smooth transitions, but the pose drifts away from the initial state (middle row, column 7). Using both x_0 and x_d results in smoother and more stable movement (bottom row).*