Improving blendshape performance for crowds with GPU and GPGPU Techniques

Timothy Costigan Anton Gerdelan Emma Carrigan Rachel McDonnell *

Graphics, Vision and Visualisation Group, Trinity College Dublin

Abstract

For real-time applications, blendshape animations are usually calculated on the CPU, which are slow to animate, and are therefore generally limited to only the closest level of detail for a small number of characters in a scene. In this paper, we present a GPU based blendshape animation technique. By storing the blendshape model (including animations) on the GPU, we are able to attain significant speed improvements over CPU-based animation. We also find that by using compute shaders to decouple rendering and animation we can improve performance when rendering a crowd animation. Further gains are also made possible by using a smaller subset of blendshape expressions, at the cost of expressiveness. However, the quality impact can be minimised by selecting this subset carefully. We discuss a number of potential metrics to automate this selection.

Keywords: facial animation, performance

Concepts: •**Computing methodologies** → **Motion capture;**

1 Introduction

In real-time rendering, facial animation is usually performed using linear blend skinning which can be readily accelerated using the Graphics Processing Unit (GPU). More realistic results can be achieved through blendshape animation, but at the cost of greater memory consumption and computational expense. Implementing blendshapes on the GPU is not straightforward, and until recently, was quite difficult due to limitations imposed by contemporary graphics hardware and drivers. As a result, most implementations have been on the CPU, such as the popular video-game Ryse Son of Rome by Crytek [Evans et al. 2014] which used corrective blendshapes at close distances. Performance on the CPU is acceptable when rendering small numbers of characters, but with large crowds performance will suffer unless we move the computation over to the highly parallel GPU.

The contributions of this paper are:

- A general purpose GPU (GPGPU) variation to shader-based blendshape animation using compute shaders and texture buffers, supporting very high complexity meshes.
- A system for discarding perceptually less important blendshapes to improve performance for crowds.
- Comparative performance evaluations of CPU and GPU blendshape techniques.
- Result-based discussion of where our blendshape animation techniques are most beneficial.

Figure 1: *A real-time rendering of 40 blend-shape animated heads using our GPU technique.*

We would like to leverage the power of commodity GPUs to allow blendshapes to be used at more than just the closest level of detail, and potentially for multiple characters in a scene. The challenges here are finding out how to arrange the data and animation processing that we need, such that it is congruent with the latest GPU rendering pipelines, and then finding a system that suits our scene for discarding processing that is less perceptually important to our rendering.

We propose a GPU-based blendshape technique which exploits relatively recent advances in GPU technology such as Texture Buffer Objects (TBO) and compute shaders. TBOs allow arbitrarily large blocks of data to be passed to the shader. Due to the highly parallel nature of the GPU, we can expect significant performance improvements over CPU based implementations. We investigate further potential speed improvements through the use of general purpose GPU programming (in the form of compute shaders) and by controlling the level of detail of animations by only processing a subset of the blendshape expressions. Compute shaders give us the advantages of using the GPU for general-purpose programming, which means we are able to compute tasks not necessarily part of the graphics pipeline, in the same fashion as CUDA and OpenCL interfaces. Compute shaders offer the additional benefit of being built into a rendering API, so are able to transfer the results of computations over to the graphics rendering pipeline within the same workflow.

2 Related Work

For high quality facial animation, the current state of the art is blendshape animation [Lewis and Anjyo 2010; Seol et al. 2011; Seol et al. 2012]. This method has seen use in many films such as The Lord of the Rings and Star Wars [Deng and Noh 2008]. Unlike with linear blend skinning, where the vertices of a mesh are

*email: Rachel.McDonnell@scss.tcd.ie

MiG '16,, October 10 - 12, 2016, Burlingame, CA, USA
ISBN: 978-1-4503-4592-7/16/10
DOI: http://dx.doi.org/10.1145/2994258.2994275

deformed based on the motions of a separate bone hierarchy [Pilgrim et al. 2007], blendshape animation uses a large number of meshes, each representing a specific expression. By linearly combining these expression *blendshapes* through equation 1, we can produce more complicated composite expressions as in Figure 2:

$$\nu_j = \sum_{k=0}^{n} \omega_k b_{kj} \tag{1}$$

where j is the current vertex index, n is the number of blendshapes, ν_j is the output vertex, ω_k is a weight between 0 and 1 determining how active the current blendshape is and b_{kj} is the input vertex of blendshape k.

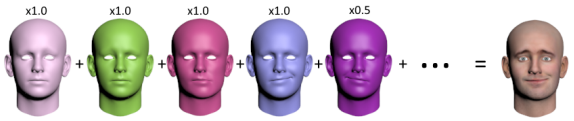

Figure 2: *Combining expression blendshapes to make a final, composite expression.*

One issue with blendshape animation is deciding what exactly these expression blendshapes will be. There is currently no consensus on the matter and no industry standard exists. However, one solution is to base these expressions on the Facial Action Coding System (FACS) of Ekman et al. [1978]. FACS defines a set of elemental facial expressions or more specifically fundamental muscle activations, known as action units (AUs). These AUs can be combined to create almost any facial expression, which makes them an ideal fit for a blendshape model.

Blendshapes are popular for realistic animation as they allow a large degree of artistic control [Seo et al. 2011]. If additional expressivity is required, one can simply add more blendshapes. Other systems such as linear blend skinning are more constrained and might require a reworking of the animation rig to facilitate new expressions. Blendshapes are not without disadvantages though. In particular, they can take a very long time to produce, as each expression must be hand crafted and requires considerably more vertices than other methods. For example, our 3D model in Figure 2 is composed of 72 blendshapes containing 43,788 vertices each, which amounts to over 3 million vertices. Movie standard blendshape models can often have hundreds of blendshapes, covering not only the basic expressions, but also ones correcting geometric anomalies when certain expressions are active.

Early attempts to perform blendshape animation on the GPU were limited. Methods based on vertex attributes were restricted to a fairly small number of blendshapes (typically 4) [Lorach 2007]. The number of blendshapes could be increased by accumulating the final blendshape over multiple passes but this involved using the CPU to drive the blendshapes. Lorach [2007] proposed an interesting buffer-template method using more modern GPU features. Similar to our technique, this method computes the final blendshape expressions on the vertex shader with the blendshape set being passed to the GPU via shader resource buffers. However there are some key differences: we perform the animation on the GPU using a texture technique similar to Dudash [2007] and we decouple the blendshape calculation from the rendering using compute shaders.

On the GPU, the number of blendshapes we can render is still limited and due to their expense, blendshapes may benefit from some form of level of detail control to improve performance (at the expense of visual quality). Geometry reducing methods could be applied such as the edge collapse method of Garland and Heckbert [1997] but maintaining correspondence between blendshapes is difficult. While Mohr and Gleicher [2003] adapted Garland and Heckbert's method to work with blendshapes, it still remains an expensive process that is only really suitable as an offline pre-process step. Also, while geometry reduction may in theory maintain expressiveness (by retaining blendshapes), it may also produce unappealing artifacts and finer expressions may be lost at lower resolutions.

A better idea may be to retain the geometric complexity of the mesh but reduce the blendshape set. This should theoretically lower the computational and memory burden of the technique but retain the resolution necessary for fine, nuanced expressions. Principal component analysis (PCA) compression is one possible method for compressing the dataset. The fact that PCA shares many theoretical similarities to blendshapes makes it an obvious candidate but according to Lewis et al. [2014] PCA gives poor compression ratios. It also changes the very nature of the model and animations, trading a dense set of intuitive expressions for a sparser but less intuitive set. By unintuitive, we mean that PCA blendshapes do not necessarily represent any recognisable expressions. Animations also change from large values on a small set of expressions to small values across the entire range of expressions. Overall, these changes make it very difficult for an animator to understand or control the animation.

Another possible method, according to Lewis et al., is to apply hierarchically semi-separable algorithms to the matrices. This replaces the denser blendshape model with a coarser sparse matrix model but such a system is quite complex to integrate into existing game engines. A potentially better idea is to remove extraneous blendshapes entirely, in order of their perceptual importance. This retains the sensible animation weights and is much easier to integrate into any game engine. Such a method can be applied easily in real-time unlike the geometry reduction discussed earlier. The buffer-template method of Lorach [2007] does something similar by only passing active blendshapes to the shader.

However, we would like to make more significant reductions by removing even active blendshapes. The issue then, is how to determine the relative importance of the facial features to minimise the impact on quality. Many general purpose metrics exist for comparing images and meshes, such as the Hausdorff distance, means squared error (MSE), structural similarity index (SSIM) or the perceptual difference metric (more details in section 4.1). These could potentially be used for ordering expressions, although it is unclear if such an order is sensible for facial animation. We implement a number of these techniques in order to determine if all or any are suitable for this purpose.

3 Method

On older GPU hardware, the most sensible technique for GPU based blendshape animation would be to pass the expressions in the form of either uniform variables or vertex attributes [Lorach 2007]. The number of uniforms or vertex attributes is quite limited however, and would likely be insufficient for detailed models with a large number of blendshapes. By using relatively recent features of the GPU we are able to overcome these limitations. Our basic method is similar to Larach [2007] but differs in how it handles the geometry and animation. The basic structure of our method can be seen in Algorithm 1.

We are able to pass the 72 blendshapes of our model to the shader by packing them inside a TBO as in Figure 3. These are one dimen-

Figure 3: *The blendshapes are serialized and stacked together in the 1-dimensional Texture Buffer Object.*

sional textures which allow extremely large collections of arbitrary data [Segal and Akeley 2012]. The maximum size of the TBO is effectively only limited by the amount of GPU memory available. For our model, we require less than 40 MB of texture memory - comparable to two uncompressed 2k textures (2048x2048x4 channels), to store all vertex blendshape and animation data. If we wanted to add normal, tangent or texture coordinate blendshapes to the TBO this would increase, but only linearly, and well within the limits of modern GPU memory, which is in the order of gigabytes.

Figure 4: *The blendshape animation encoded in a 32 bit floating-point texture. Each row contains the animated weight of a specific blendshape over time. We can see a group of blendshapes activate in the middle (bright white).*

As the geometry resides on and is manipulated by the GPU, the graphics driver does not need to be stalled to transfer geometry to the video memory. In addition to storing the blendshapes on the GPU, using a similar technique to [Dudash 2007], we also store the entire animation on the GPU as well. This was achieved by packing the animation values into a m by n texture, where m is the number of blendshapes, and n is the number of frames as in Figure 4. To ensure precision is preserved, a 32 bit floating-point texture is used. Texture components are usually represented using only 8 bits, which would severely quantise our animation data. Moving the animation onto the GPU removes the need to calculate and pass the animation weights to the shader every frame, the cost of which we find to be significant. Moving these calculations to the vertex shader further reduces the number of operations that might stall the GPU driver and frees the CPU for other purposes.

To reduce the number of iterations (and therefore improve performance), we have modified our blendshape shader to be able to return early, similar to the method of Larach [2007]. This can be seen in Algorithm 1 where, inside the `for` loop, a conditional checks against the number of blendshapes, N_{Bs}. If this many iterations have been computed, then the shader returns early. This smaller number of blendshapes is specified by the user. This effectively trades animation quality for animation performance. As can be seen later in Section 6, this is necessary if we want practical real-time performance in large scenes. Such a system may appear initially unattractive, as a naïve implementation will result in a reduction in quality that can be severe. If, however, an importance order is established for each expression, the effect on quality can be minimised. This will be discussed in more detail in Section 4.

3.1 Compute Shader

A disadvantage of performing all blendshape related calculations on the GPU through the vertex shader, is that animation and rendering become inextricably linked. This is because everything now resides within the standard graphics rendering pipeline. This means that

Algorithm 1 Blendshape Pseudo-code

N_{Bs} = Blendshape Limit
V_{Idx} = Current Vertex Index
T_{Anim} = Animation Texture
T_{Bs} = Blendshape TBO
N_V = Number of Vertices per blendshape
t = Current Frame
$VertexPos$ = Initial neutral vertex position at index V_{Idx}

for $i = 0$ **to** $MAXBLENDSHAPES$ **do**
  $Weight = T_{Anim}[i,t]$
  if $i < N_{Bs}$ **then**
    $T_{Idx} = i * N_V * 3 + V_{Idx} * 3$ //TBO index
    $TempPos$ = (0,0,0)
    $TempPos$.x = $T_{Bs}[T_{Idx}]$
    $TempPos$.y = $T_{Bs}[T_{Idx} + 1]$
    $TempPos$.z = $T_{Bs}[T_{Idx} + 2]$
    $VertexPos = VertexPos + TempPos * Weight$
  else
    Escape Loop
  end if
end for

all calculations will need to be repeated every draw call whether they are required or not. A CPU based solution does not share this limitation.

If we render multiple characters with different animation frames, repeating the calculations is not a problem but in a crowd rendering context this does limit us. This is because the perceived size of a crowd can be increased by interspersing duplicates of a small set of unique characters throughout the scene [McDonnell et al. 2008]. If we use our current GPU based system there is no performance advantage to this technique.

It is possible though to decouple animation and rendering while still keeping all calculations on the GPU by using GPGPU programming in the form of compute shaders. These programs can be executed on the GPU, accessing all the same texture and vertex data as our normal method but operate independent of rendering. We are even able to run the same method as Algorithm 1 with minimal alteration. Compute shaders were first added in OpenGL 4.3 [Segal and Akeley 2012], two years after DirectX 11.

3.2 CPU

For comparison, we also implement a basic CPU blendshape technique as most current implementations of blendshape animation are CPU based. It should also be stated that CPU based methods do have some advantages. They are invariably easier to implement and allow for a much simpler graphics pipeline which might be important if GPU rather than CPU time is at a premium.

If we are rendering small numbers of faces, real-time performance can easily be achieved on the CPU. For larger numbers of characters, the disadvantages become apparent. As the geometry must be recalculated on the CPU, it must also be passed to the GPU every frame which can be very costly as the GPU stalls while the transfer is underway.

4 Level of detail

As discussed in Section 3, our method, similar to Lorach [2007], allows the user to reduce the number of blendshapes being computed by the vertex shader, and in our case compute shader, to improve

performance. Lorach [2007] only removed inactive blendshapes, ensuring the animations would be identical. Our method goes further and sorts the blendshapes in terms of importance. By removing less important expressions first, we can make much deeper performance improvements while minimising the effect on quality.

4.1 Ordering

We investigated using a number of general purpose metrics for ranking or ordering blendshapes by perceptual importance. Every metric was used to compare each blendshape expression to the neutral version of our model's face (all blendshape weights set to zero). These metrics can roughly be classified as either geometry or image based. The geometry based metrics only consider the vertex positions of the various expressions. They are not influenced by any rendering information such as texture or lighting. In contrast, the image based metrics only consider the final rendered image and are heavily influenced by texture and lighting.

As we will discuss below, each metric has advantages and disadvantages. The results are broadly similar although there are differences for certain expressions particularly in the eyebrow region. All methods also give relatively low importance values to eye blinks. In general, we found that these metrics gave acceptable results for small to moderate blendshape reduction but some manual intervention was required for larger reductions.

4.1.1 Geometry Based : Mean Squared Error

MSE is a fairly simplistic metric which compares two objects on a vertex by vertex basis, based on the equation below:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{2}$$

where n is the number of vertices, $\hat{Y}_i$ is a neutral expression vertex, and Y_i is a vertex of the current expression being evaluated. For our model, we found that this method gave high importance values to mouth and lip expressions but low values to eye and brow expressions.

4.1.2 Geometry : Hausdorff Distance

The Hausdorff distance is a more robust metric that is commonly used to compare the differences between two different meshes based on the following equation [Aspert et al. 2002]:

$$H = max_{a\in A}\{min_{b\in B}\{d(a,b)\}\} \tag{3}$$

where A and B are the meshes being compared, a and b are vertices of their respective meshes and d is the Euclidean distance. Similar to the geometry based MSE, we found this metric gave high values to the mouth region and lower values to the eye region. It gave higher values to the eyebrows though than most methods.

4.1.3 Image Based : Mean Squared Error

The image based variant of mean squared error metric is the same as the geometric version although it operates on a pixel by pixel basis.

$$MSE = \frac{1}{wh}\sum_{y=1}^{h}\sum_{x=1}^{w}(I_a(x,y) - I_b(x,y))^2 \tag{4}$$

Where I_a and I_b represent two 2d input images with dimensions w and h, and pixel co-ordinates given by x and y, respectively. In our tests, this metric gave high importance values to the mouth and brow region of our model. Blinks were also not too low. This method can be a little too sensitive though. For example, minor asymmetrical differences between the left and right of the face texture can result in different importance values for otherwise identical expressions.

4.1.4 Image Based : Structural Similarity Index

The structural similarity index metric (SSIM) tries to model the response of the human vision system [Wang et al. 2004]. We found this method gives identical importance values to symmetrical expressions unlike the image based MSE. In terms of importance values, we found the brow expressions of our model were given relatively high values compared with the other metrics although cheek and blink expressions were relatively low.

4.1.5 Image Based : Perceptual Difference

The perceptual difference metric, created by Yee [2004] is an attempt to model the human vision system to an even higher degree of accuracy than Wang et al. [2004]. We found this method was sensitive to certain subtle expressions of our model such as cheek motions.

5 Test Data

To acquire our data for testing, we captured a volunteer performing 6 unique sentences from the TCD-TIMIT [2015] dataset. We selected this because its sentences are very dense phonetically, which should help ensure that more mouth blendshapes are activated. Facial motion was captured using a tripod mounted *GoPro* camera at 120 fps with a resolution of 1920x1080. This motion was tracked and retargeted to our 3D model (Figure 2), which has 72 high quality blendshape expressions closely matching the FACS action units, using the professional pose-based solution by *FaceWare* [Faceware 2016].

6 Results

Figure 5 gives our measured timing results. Our results are based on a sample size of $n = 100$ measurements, and error bars shown are plus and minus 1 standard error from the mean. All plots have error calculated, but most results have negligible variation. The columns show results from 3 scene configurations:

- 1 head rendered with unique blendshape animation frames per head.
- 100 heads rendered with unique blendshape animation frames per head.

This means that, in the case of 100 heads in Figure 5, each of our timings also includes the cost of computing a set of unique blendshape animations for each head - no instancing or cloning is used. The rows show results for each scene for two timers:

- GPU timer queries to measure the time to render the scene on the graphics device (in milliseconds).
- Round-trip frame time on the CPU (in milliseconds), which includes the cost of drawing as well as CPU-side driver overheads, and associated uniform variable updates.

All results are measured on a 64-bit 6-core Intel Core i7 3.5GHz CPU with a Nvidia GeForce GTX 960 GPU, using Windows 10 64-bit driver version 362.00. We explicitly disable rasterisation before drawing, so that our measurements only consider the shader

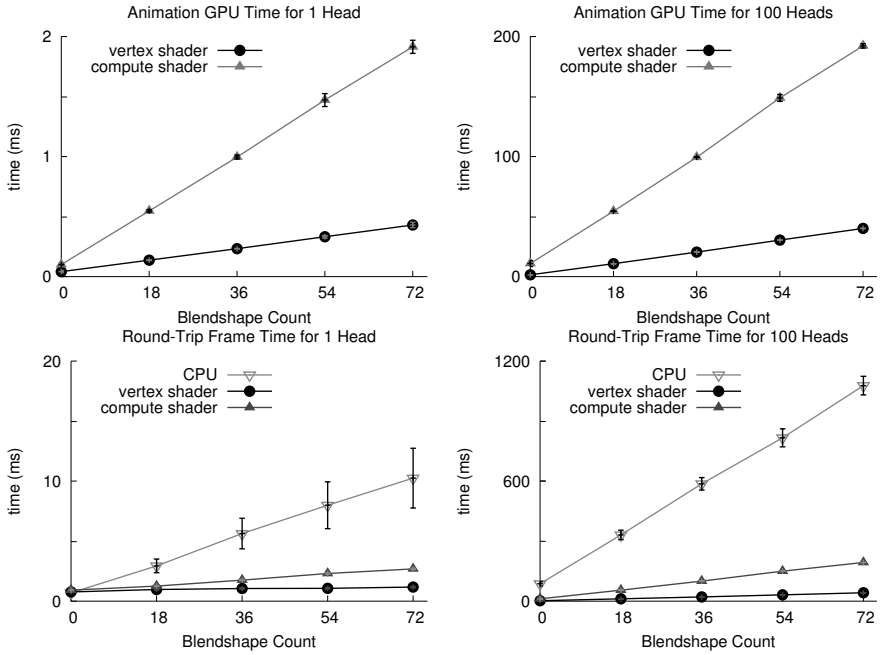

Figure 5: *Plots compare the basic CPU method, the vertex shader method, and the compute shader method. The top row gives GPU times for rasterisation-disabled drawing for 1 and 100 heads, each head with unique blendshape weights. The bottom row gives corresponding times on the CPU.*

processing costs related to animation, and not any overheads created by rasterisation and fragment shading stages of the pipeline.

In all tested techniques, we observe a linear decrease in drawing time on the GPU as blendshapes are discarded. We note, in particular, that in this general purpose case the vertex shader-based animation technique is always faster than compute-shader and CPU-based animation techniques. All of the techniques have a linear cost increase as both the number of blendshapes computed, and the number of animated heads drawn increases.

We also measured for the case of cloning or reusing animation sequences within a crowd. This means that animations are computed once, and shared with subsequently drawn heads. In this special case we see compute shader animations processing many times faster than other methods on the GPU (Figure 6 *(top)*), and this gives an interesting case to leverage general computing shaders to performance advantage over the traditional pipeline. The CPU round-trip time (Figure 6 *(bottom)*) is negligible for both shader-based implementations.

7 Conclusions and Future Work

We have demonstrated a technique for rendering large numbers of blendshape characters in real-time using the GPU. By storing all geometry and animation information on the GPU as textures, we were able to minimise expensive graphics driver computational overheads inherent in any CPU technique. Our method when coupled with compute shaders, makes larger, fully blendshape animated crowds a possibility.

In terms of performance, we observed that the vertex shader technique was significantly faster than the CPU (or even compute shader methods in most configurations). We found that the compute shader has a much more costly overhead associated with its use, although performance was superior to the CPU technique. The compute shader method has the advantage of decoupling render-

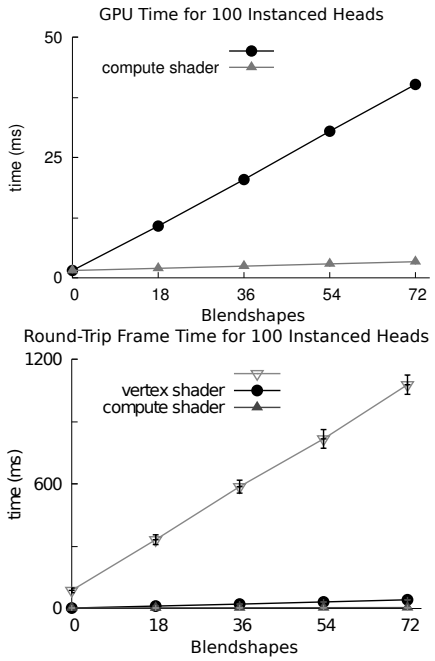

Figure 6: *The special case of instanced rendering for crowds (sharing weights) on the GPU (top), and round-trip frame time on the CPU (bottom). We observe performance advantage in leveraging generic computing in shaders here.*

ing from animation. This means in certain circumstances, such as crowd rendering where instancing is appropriate, significant speed improvements could be gained by reducing the number of compute shader calls, as can be seen in Figure 5.

Performance improvements were found to increase linearly with respect to the number of blendshapes computed, with both vertex and compute shader techniques. Using a smaller subset of blendshapes does, however, reduce the expressiveness of the model as full expressions are being discarded. As a result, using a lower number of blendshapes without ordering gave inconsistent results for our model. We found though that by using sensibly ordered expressions, we were able to reduce the impact on animation quality and achieve much higher blendshape reduction levels. We expect that these reduced blendshapes would be used in less salient background characters.

We performed an ad-hoc assessment of a variety of standard geometric and vision based metrics to help establish a sensible automatic ordering for our level of detail method. As can be seen in the supplemental video, reductions of up to 50% can be achieved for our dataset examples with little manual intervention when using all 5 metrics. At a more significant reduction level of 75%, lip sync is lost when using the image based MSE and perceptual difference metrics. Lip sync was mostly maintained, however, with the geometric MSE, SSIM and Hausdorff distance metrics. We found that all methods gave small yet perceptually significant expressions such as blinks low importance values, so some manual intervention is likely required depending on the animation.

Our method considers the perceived prominence of individual blendshapes in the rig for blendshape reduction, but it is possible that further reductions would be possible depending on the animation type (e.g., conversation, emotion, etc.). A thorough perceptual evaluation could extend this work to quantify which metrics detailed in section 4.1 are perceptually most effective for different animation types, with special consideration for use in a crowd rendering environment.

Acknowledgements

This research was funded by Science Foundation Ireland under the ADAPT Centre for Digital Content Technology (Grant 13/RC/2106) and the Game Face (13/CDA/2135) project.

References

ASPERT, N., SANTA CRUZ, D., AND EBRAHIMI, T. 2002. MESH: measuring errors between surfaces using the Hausdorff distance. In *ICME (1)*, 705–708.

DENG, Z., AND NOH, J. 2008. Computer facial animation: A survey. In *Data-driven 3D facial animation*. Springer, 1–28.

DUDASH, B. 2007. Skinned instancing. Tech. rep., NVidia.

EKMAN, P., AND FRIESEN, W. V. 1978. *Facial action coding system*. Consulting Psychologists Press, Stanford University.

EVANS, C., MARTINSSON, L., AND HERFORT, S. 2014. Building an empire: asset production in Ryse. In *ACM SIGGRAPH 2014 Courses*, 18.

FACEWARE, 2016. Faceware Technologies Inc. http://facewaretech.com/.

GARLAND, M., AND HECKBERT, P. S. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 209–216.

HARTE, N., AND GILLEN, E. 2015. TCD-TIMIT: An audio-visual corpus of continuous speech. *IEEE Transactions on Multimedia 17*, 5, 603–615.

LEWIS, J., AND ANJYO, K.-I. 2010. Direct manipulation blendshapes. *IEEE Computer Graphics and Applications*, 4, 42–50.

LEWIS, J. P., ANJYO, K., RHEE, T., ZHANG, M., PIGHIN, F., AND DENG, Z. 2014. Practice and Theory of Blendshape Facial Models. In *Eurographics 2014 - State of the Art Reports*.

LORACH, T. 2007. *DirectX 10 Blend Shapes: Breaking the Limits*. GPU Gems 3, Addison-Wesley Professional, ch. 3, 53–67.

MCDONNELL, R., LARKIN, M., DOBBYN, S., COLLINS, S., AND O'SULLIVAN, C. 2008. Clone attack! perception of crowd variety. *ACM Transactions on Graphics (TOG) 27*, 3, 26.

MOHR, A., AND GLEICHER, M. 2003. Deformation sensitive decimation. Tech. rep., University of Wisconsin Graphics Group.

PILGRIM, S., STEED, A., AND AGUADO, A. 2007. Progressive skinning for character animation. *Computer Animation and Virtual Worlds 18*, 4-5, 473–481.

SEGAL, M., AND AKELEY, K. 2012. The OpenGL Graphics System: A Specification (Version 4.3 (Core Profile)). Tech. rep., Khronos Group.

SEO, J., IRVING, G., LEWIS, J. P., AND NOH, J. 2011. Compression and Direct Manipulation of Complex Blendshape Models. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, 164:1–164:10.

SEOL, Y., SEO, J., KIM, P. H., LEWIS, J. P., AND NOH, J. 2011. Artist friendly facial animation retargeting. In *ACM Transactions on Graphics (TOG)*, 162.

SEOL, Y., LEWIS, J., SEO, J., CHOI, B., ANJYO, K., AND NOH, J. 2012. Spacetime expression cloning for blendshapes. *ACM Transactions on Graphics (TOG) 31*, 2, 14.

WANG, Z., BOVIK, A. C., SHEIKH, H. R., AND SIMONCELLI, E. P. 2004. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on 13*, 4, 600–612.

YEE, H. 2004. Perceptual metric for production testing. *Journal of Graphics Tools 9*, 4, 33–40.