Practical and Realistic Facial Wrinkles Animation

Jorge Jimenez, Jose I. Echevarria,
Christopher Oat and Diego Gutierrez

Virtual characters in games are becoming more and more realistic, with recent advances for instance in the fields of skin rendering [d'Eon and Luebke 07, Hable et al. 09, Jimenez and Gutierrez 10] or behavior-based animation [NaturalMotion 05]. To avoid lifeless representations and help the user engage in the action, more and more sophisticated algorithms are being devised that capture subtle aspects of the appearance and motion of these characters. Unfortunately, facial animation and the emotional aspect of the interaction have not been traditionally pursued with the same intensity. We believe this is an important missing aspect in games, especially given the current trend of story-driven AAA games and their movie-like real-time cutscenes.

Without even realizing it, we often depend on the subtleties of facial expression to give us important contextual cues about what someone is saying, thinking or feeling. For example, a wrinkled brow can indicate surprise, while a furrowed brow may indicate confusion or inquisitiveness. In the mid-1800s, a French neurologist named Guillaume Duchenne performed experiments that involved applying electric simulation to his subject's facial muscles. Duchenne's experiments allowed him to map which facial muscles were used for different facial expressions. One interesting fact that he discovered was that smiles resulting from true happiness not only utilize the muscles of the mouth, but also those of the eyes. It is this subtle but important additional muscle movement that distinguishes a genuine happy smile from an inauthentic or sarcastic smile. What we learn from this is that facial expressions are complex and sometimes subtle, but extraordinarily important in conveying meaning and intent. In order to allow artists to create realistic, compelling characters we must allow them to harness the power of subtle facial expression.

We present a method to add expressive animated wrinkles to characters, helping enrich stories through subtle visual cues. Our system allows the animator to independently blend multiple wrinkle maps across regions of a character's face. We demonstrate how combining our technique with state-

Without wrinkles With wrinkles

Figure 1.1. The same scene without and with animated facial wrinkles. Adding them helps to increase visual realism and conveys mood to the character.

of-the-art real-time skin rendering can produce stunning results that bring out the personality and emotional state of a character (see Figures 1.1 and 1.2).

This enhanced realism has little performance impact. In fact our implementation has a memory footprint of just 96 KB. Performance wise, the execution time of our shader is 0.31 ms, 0.1 ms and 0.09 ms on a low-end GeForce 8600GT, mid-range GeForce 9800GTX+ and mid-high range GeForce 295GTX respectively. Furthermore, it is simple enough to be easily added to existing rendering engines without requiring drastic changes, even allowing to reuse existing bump/normal textures, as our technique builds on top of them.

1.1 Background

Bump maps and normal maps are well known techniques for adding the illusion of surface features to otherwise coarse, undetailed surfaces. The use of normal maps to capture the facial detail of human characters is considered standard practice for the past several generations of real-time rendering applications. However, using static normal maps unfortunately does not accurately represent the dynamic surface of an animated human face. In order to simulate dynamic wrinkles, one option is to use length-preserving geometric constraints along with artist-placed wrinkle features to dynami-

Figure 1.2. This figure shows our wrinkle system in action for a complex facial expression composed of multiple, simultaneous blend shapes.

cally create wrinkles on animated meshes [Larboulette and Cani 04]. Since this method actually displaces geometry, the underlying mesh must be sufficiently tessellated to represent the finest level of wrinkle detail. A dynamic facial wrinkle animation scheme presented recently [Oat 07] employs two wrinkle maps (one for stretch poses and one for compress poses) and allows them to be blended to independent regions of the face using artist animated weights along with a mask texture. We build upon this

technique, demonstrating how to dramatically optimize the memory requirements. Furthermore, our technique allows to easily include more than two wrinkle maps when needed, as we no longer map negative and positive values to different textures.

1.2 Our Algorithm

The core idea of this technique is the addition of wrinkle normal maps on top of the base normal maps and blend shapes (see Figure 1.3, *left* and *center* for example maps). For each facial expression, wrinkles are selectively applied by using weighted masks (see Figure 1.3, *right*, and Table 1.1 for the mask and weights used in our examples). This way, the animator is able to manipulate the wrinkles on a per-blend-shape basis, allowing art-directed blending between poses and expressions. We store a wrinkle mask per channel of a RGBA texture; this way we can store up to four zones per texture. As our implementation uses 8 zones, we only require storing and accessing two textures. Note that when the contribution of multiple blend shapes in a zone exceeds a certain limit, artifacts can appear in the wrinkles. In order to avoid this problem, we clamp the value of the summation to the $[0, 1]$ range.

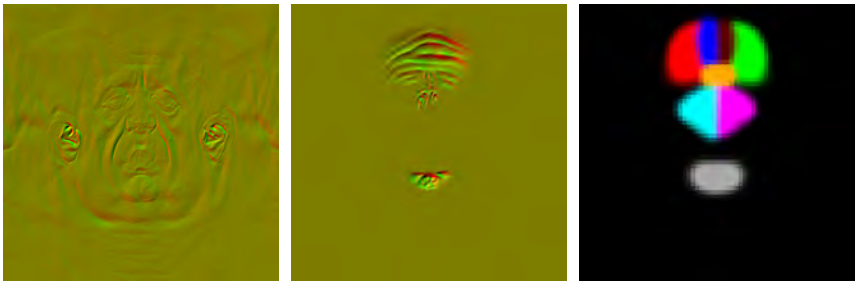
While combining various displacement maps consists of a simple sum, combining normal maps involves complex operations that should be avoided in a time-constrained environment like a game. Thus, in order to combine the base and wrinkle maps a special encoding is used: partial derivative normal maps [Acton 08]. It has two advantages over the conventional normal map encoding: a) instead of reconstructing the z value of a normal, we just have to perform a vector normalization, saving valuable GPU cycles; b) more important for our purposes, the combination of various partial derivative normal maps is reduced to a simple sum, similar to combining displacement maps.

This encoding must be run as a simple pre-process. Converting a conventional normal $n = (n_x, n_y, n_z)$ to a partial derivative normal $n' = (n'_x, n'_y, n'_z)$ is done by using the following equations:

$$n'_x = \frac{n_x}{n_z} \quad n'_y = \frac{n_y}{n_z}$$

In runtime, reconstructing a single partial derivative normal n' to a conventional normal $\hat{n}$ is done as follows:

$$\begin{aligned} n &= (n'_x, n'_y, 1) \\ \hat{n} &= \frac{n}{\|n\|} \end{aligned}$$



Base Map Wrinkle Map Mask Map

Figure 1.3. The wrinkle map is selectively applied on top of the base normal map by using a wrinkle mask. The usage of partial derivative normal maps reduces this operation to a simple addition. The yellowish look is due to the encoding and storage in the R and G channels that this technique employs. Wrinkle zone colors in the mask do not represent the actual channels of the mask maps, they are put together just for visualization purposes.

	Red	Green	Blue	Brown	Cyan	Magenta	Orange	Gray
Joy	1.0	1.0	0.2	0.2	0.0	0.0	0.0	0.0
Surprise	0.8	0.8	0.8	0.8	0.0	0.0	0.0	0.0
Fear	0.2	0.2	0.75	0.75	0.3	0.3	0.0	0.6
Anger	-0.6	-0.6	-0.8	-0.8	0.8	0.8	1.0	0.0
Disgust	0.0	0.0	-0.1	-0.1	1.0	1.0	1.0	0.5
Sad	0.2	0.2	0.75	0.75	0.0	0.0	0.1	1.0

Table 1.1. Weights used for each expression and zone (see color meaning in the mask map of Figure 1.3).

Note that in the original formulation of partial derivative normal mapping there is a minus sign both in the conversion and reconstruction phases; removing it from both steps allows to obtain the same result with the additional advantage of saving another GPU cycle.

Then, combining different partial derivative normal maps consists on a simple summation of their (x, y) components before the normalization step. As Figure 1.3 reveals, expression wrinkles are usually low frequency. Thus, we can reduce map resolution to spare storage and lower bandwidth consumption, without visible loss of quality. Calculating the final normal map is therefore reduced to a summation of weighted partial derivative normals (see Listing 1.1).

A problem with facial wrinkle animation is the modeling of compound expressions, where the resulting wrinkles come from the interactions between the basic expressions they are built upon. For example, if we are surprised, the Frontalis muscle contracts the skin producing wrinkles in the forehead. If we then suddenly became angry, the Corrugator muscles would

```
float3 WrinkledNormal(Texture2D<float2> baseTex,
                      Texture2D<float2> wrinkleTex,
                      Texture2D maskTex[2],
                      float4 weights[2],
                      float2 texcoord) {
  float3 base;
  base.xy = baseTex.Sample(AnisotropicSampler16, texcoord).gr;
  base.xy = -1.0 + 2.0 * base.xy;
  base.z = 1.0;

  #ifdef WRINKLES
  float2 wrinkles = wrinkleTex.Sample(LinearSampler,
                                      texcoord).gr;
  wrinkles = -1.0 + 2.0 * wrinkles;

  float4 mask1 = maskTex[0].Sample(LinearSampler, texcoord);
  float4 mask2 = maskTex[1].Sample(LinearSampler, texcoord);
  mask1 *= weights[0];
  mask2 *= weights[1];

  base.xy += mask1.r * wrinkles;
  base.xy += mask1.g * wrinkles;
  base.xy += mask1.b * wrinkles;
  base.xy += mask1.a * wrinkles;
  base.xy += mask2.r * wrinkles;
  base.xy += mask2.g * wrinkles;
  base.xy += mask2.b * wrinkles;
  base.xy += mask2.a * wrinkles;
  #endif

  return normalize(base);
}
```

Listing 1.1. HLSL code of our technique. We are using a linear instead of an anisotropic sampler for the wrinkle and mask maps because the low-frequency nature of their information does not require higher quality filtering. This code is a more readable version of the optimized code found in the web material.

be triggered, which would expand the skin in the forehead, thus causing the wrinkles to disappear. To be able to model this kind of interactions, we let mask weights take negative values, allowing to cancel each other. Figure 1.5 illustrates this particular situation.

1.2.1 Alternative: using normal map differences

An alternative to the usage of partial derivative normal maps for combining normal maps, is to store differences between the base and each of the expression wrinkle maps (see Figure 1.4, *right*), in a similar way to how blend shape interpolation is usually performed. As differences may contain

negative values, we perform a scale and bias operation so that all values fall in the $[0, 1]$ range, enabling its storage in regular textures:

$$d(x, y) = 0.5 + 0.5 \cdot (w(x, y) - b(x, y)),$$

where $w(x, y)$ is the normal at pixel (x, y) of the wrinkle map, and $b(x, y)$ is the corresponding value from the base normal map. When DXT compression is used for storing the differences map, it is recommended to renormalize the resulting normal after adding the delta, in order to alleviate the artifacts caused by the compression scheme (see web material for the corresponding listing).

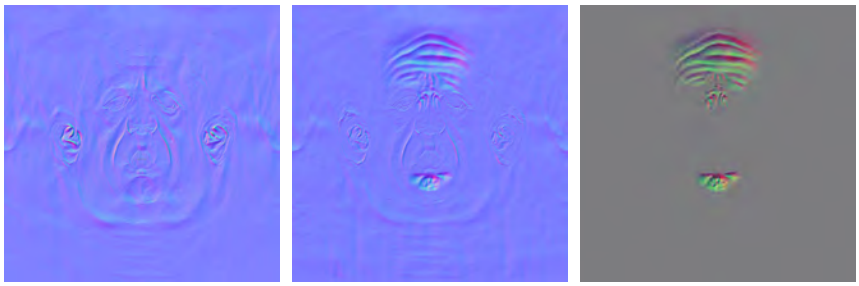


Figure 1.4. We calculate a wrinkle difference map by subtracting the base normal map from the wrinkle map. In runtime, the wrinkle difference map is selectively added on top of the base normal map by using a wrinkle mask (see Figure 1.3, *right*, for the mask). The grey color of the image on the right is due to the bias and scale introduced when computing the difference map.

Partial derivative normal mapping has the following advantages over the differences approach:

- It can be a little bit faster as it saves one GPU cycle when reconstructing the normal and also allows to only add two-component normal derivatives instead of a full (x, y, z) difference; these two-component additions can be done two at once in only one cycle. This translates to a measured performance improvement of $1.12x$ in the GeForce 8600GT, whereas we have not observed any performance gain in either the GeForce 9800GTX+ nor in the GeForce 295GTX .

- It only requires two channels to be stored vs. the three channels required for the differences approach. This implies higher quality as 3Dc can be used to compress the wrinkle map for the same memory cost.

On the other hand, the differences approach has the following advantages over the partial derivative normal mapping approach:

- It uses standard normal maps, which may be important if this cannot be changed in the production pipeline.
- Partial derivative normal maps cannot represent anything that falls outside of a 45 degree cone around $(0, 0, 1)$. Nevertheless, in practice, this problem proved to have little impact on the quality of our renderings.

The suitability of each approach will depend on both the constraints of the pipeline and the characteristics of the art assets.

1.3 Results

For our implementation we used DirectX 10, but the wrinkle animation shader itself could be easily ported to DirectX 9. However, to circumvent the limitation that only four blend shapes can be packed into per-vertex attributes at once, we used the DirectX 10 stream out feature, which allows us to apply an unlimited number of blend shapes using multiple passes [Lorach 07]. The base normal map has a resolution of 2048×2048, whereas the difference wrinkle and mask maps have a resolution of 256×256 and 64×64 respectively, as they only contain low-frequency information. We use 3Dc compression for the base and wrinkle maps, and DXT for the color and mask maps. The high-quality scanned head model and textures were kindly provided by XYZRGB, with the wrinkle maps created manually, adding the missing touch to the photorealistic look of the images. We used a mesh resolution of 13063 triangles, mouth included, which is a little step ahead from current generation of games; however, as current high-end system become mainstream, it will be more common to see such high polygon counts, specially in cinematics.

To simulate the subsurface scattering of the skin, we use the recent screen-space approach [Jimenez and Gutierrez 10,Jimenez et al. 10b], which transfers computations from texture space to screen space by modulating a convolution kernel according to depth information. This way, the simulation is reduced to a simple post-process, independent of the number of objects in the scene and easy to integrate in any existing pipeline. Facial color animation is achieved using a recently proposed technique [Jimenez et al. 10a], which is based on in vivo melanin and hemoglobin measurements of real subjects. Another crucial part of our rendering system is the Kelemen/Szirmay-Kalos model, which provides realistic specular reflec-

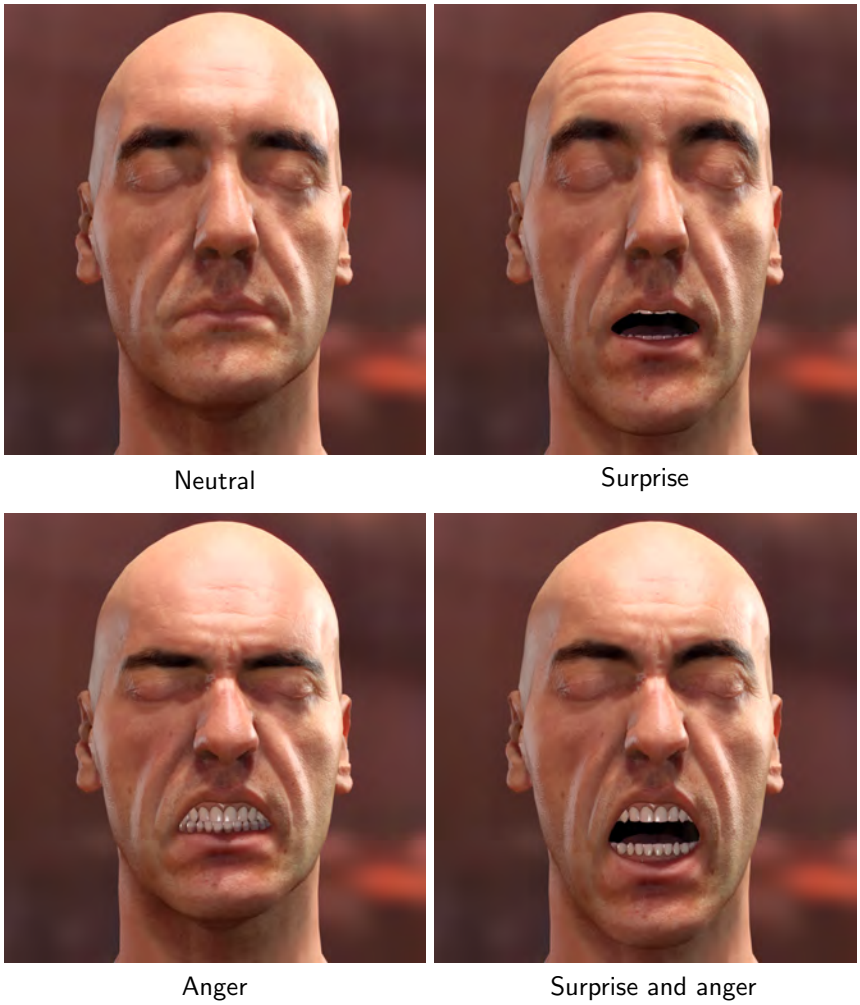


Figure 1.5. The net result of applying both surprise and anger expressions on top of the neutral pose is a wrinkleless forehead. In order to accomplish this, we use positive and negative weights in the forehead wrinkle zones, for the surprise and angry expressions respectively.

tions in real-time [d'Eon and Luebke 07]. Additionally, we use the recently introduced Filmic tone mapper [Hable 10], which yields really crisp blacks.

For the head shown in the images, we have not created wrinkles for the zones corresponding to the cheeks because the model is tessellated enough in this zone, allowing to produce geometric deformations directly on the

	Shader execution time
GeForce 8600GT	0.31 ms
GeForce 9800GTX+	0.1 ms
GeForce 295GTX	0.09 ms

Table 1.2. Performance measurements for different GPUs. The times shown correspond specifically to the execution of the code of the wrinkles shader.

Figure 1.6. Closeups showing the wrinkles produced by Nasalis (nose), Frontalis (forehead) and Mentalis (chin) muscles.

blend shapes.

Figure 1.6 shows different closeups that allow appreciating the wrinkles added in detail. Figure 1.7 depicts a sequence showcasing the blending between compound expressions, illustrating how adding facial wrinkle animation boosts realism and adds mood to the character (frames taken from the movie included in the web material).

Table 1.2 shows the performance of our shader using different GPUs, from the low-end GeForce 8600GT to the high-end GeForce 295GTX. An in-depth examination of the compiled shader code reveals that the wrinkle shader add a per-pixel arithmetic instruction/memory access count of 9/3. Note that animating wrinkles is mostly useful for near to medium distances; for far distances it can be progressively disabled to save GPU cycles. Besides, when similar characters share the same (u, v) arrangement, we can reuse the same wrinkles improving further the usage of memory resources.

1.4 Discussion

From direct observation of real wrinkles, it may seem natural to think that shading could be enhanced by using techniques like ambient occlusion or parallax occlusion mapping [Tatarchuk 07]. However, we have found that

wrinkles exhibit very little to no ambient occlusion, unless the parameters used for its generation are pushed beyond its natural values. Similarly, self-occlusion and self-shadowing can be thought to be an important feature when dealing with wrinkles, but in practice we have found that the use of parallax occlusion mapping is most of the times unnoticeable, for the specific case of facial wrinkles.

Furthermore, our technique allows the incorporation of additional wrinkle maps, like the lemon pose used in [Oat 07], which allows to stretch wrinkles already found in the neutral pose. However, we decided not to use them because they had little impact in the expressions we selected for this particular character model.

1.5 Conclusion

Compelling facial animation is an extremely important and challenging aspect of computer graphics. Both games and animated feature films rely on convincing characters to help tell a story and a critical part of character animation is the character's ability to use facial expression. We have presented an efficient technique for achieving animated facial wrinkles for real-time character rendering. When combined with traditional blend-target morphing for facial animation, our technique can produce very compelling results that enable virtual characters to be much more expressive in both their actions and dialog. Our system requires very little texture memory and is extremely efficient, enabling true emotional and realistic character renderings using technology available in widely adopted PC graphics hardware and current generation game consoles.

1.6 Acknowledgements

Jorge would like to dedicate this work to his eternal and most loyal friend Kazán. We would like to thank Belen Masia for her very detailed review and support, Wolfgang Engel for his editorial efforts and ideas to improve the technique, and Xenxo Alvarez for helping create the different poses. This research has been funded by a Marie Curie grant from the Seventh Framework Programme (grant agreement no.: 251415), the Spanish Ministry of Science and Technology (TIN2010-21543) and the Gobierno de Aragón (projects OTRI 2009/0411 and CTPP05/09). Jorge Jimenez was additionally funded by a grant from the Gobierno de Aragon. The authors would also like to thank XYZRGB Inc. for the high-quality head scan.

Bibliography

[Acton 08] Mike Acton. "Ratchet and Clank Future: Tools of Destruction Technical Debriefing." Technical report, Insomniac Games, 2008.

[d'Eon and Luebke 07] Eugene d'Eon and David Luebke. "Advanced Techniques for Realistic Real-Time Skin Rendering." In *GPU Gems 3*, edited by Hubert Nguyen, Chapter 14, pp. 293–347. Addison Wesley, 2007.

[Hable et al. 09] John Hable, George Borshukov, and Jim Hejl. "Fast Skin Shading." In *ShaderX*^{7}, edited by Wolfgang Engel, Chapter 2.4, pp. 161–173. Charles River Media, 2009.

[Hable 10] John Hable. "Uncharted 2: HDR Lighting." Game Developers Conference, 2010.

[Jimenez and Gutierrez 10] Jorge Jimenez and Diego Gutierrez. "Screen-Space Subsurface Scattering." In *GPU Pro*, edited by Wolfgang Engel, Chapter 5.7. A.K. Peters, 2010.

[Jimenez et al. 10a] Jorge Jimenez, Timothy Scully, Nuno Barbosa, Craig Donner, Xenxo Alvarez, Teresa Vieira, Paul Matts, Veronica Orvalho, Diego Gutierrez, and Tim Weyrich. "A Practical Appearance Model for Dynamic Facial Color." *ACM Transactions on Graphics* 29:5.

[Jimenez et al. 10b] Jorge Jimenez, David Whelan, Veronica Sundstedt, and Diego Gutierrez. "Real-Time Realistic Skin Translucency." *IEEE Computer Graphics and Applications* 30:4 (2010), 32–41.

[Larboulette and Cani 04] C. Larboulette and M. Cani. "Real-Time Dynamic Wrinkles." In *Proc. of the Computer Graphics International*, pp. 522–525. IEEE Computer Society, 2004.

[Lorach 07] T. Lorach. "DirectX 10 Blend Shapes: Breaking the Limits." In *GPU Gems 3*, edited by Hubert Nguyen, Chapter 3, pp. 53–67. Addison Wesley, 2007.

[NaturalMotion 05] NaturalMotion. "Dynamic Motion Synthesis.", 2005.

[Oat 07] Christopher Oat. "Animated wrinkle maps." In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pp. 33–37, 2007.

[Tatarchuk 07] Natalya Tatarchuk. "Practical Parallax Occlusion Mapping." In *ShaderX*^{5}, edited by Wolfgang Engel, Chapter 2.3, pp. 75–105. Charles River Media, 2007.

Figure 1.7. Transition between various expressions. Having multiple mask zones for the forehead wrinkles allows their shape to change according to the animation.