Chapter 4

Animated Wrinkle Maps

Christopher Oat[6]

AMD

Figure 1. *Ruby wrinkles her brow and purses her lips to express disapproval in the real-time short "Ruby: Whiteout".*

4.1 Abstract

An efficient method for rendering animated wrinkles on a human face is presented. This method allows an animator to independently blend multiple wrinkle maps across multiple

[6] email: chris.oat@amd.com

regions of a textured mesh such as the female character shown in Figure 1. This method is both efficient in terms of computation as well as storage costs and is easily implemented in a real-time application using modern programmable graphics processors.

4.2 Introduction

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 an important part of character animation is the character's ability to use facial expression. Without even realizing it, we often depend on the subtleties of facial expression to give us important contextual cues about what someone is saying to us. For example a wrinkled brow can indicate surprise while a furrowed brow may indicate confusion or inquisitiveness.

In order to allow artists to create realistic, compelling characters we must allow them to harness the power of subtle facial expression. The remainder of these notes will describe a technique for artist controllable wrinkles. This technique involves compositing multiple wrinkle maps using a system of masks and artist animated weights to create a final wrinkled normal map that is used to render a human face.

4.3 Wrinkle Maps

Wrinkle maps are really just bump maps that get added on top of a base normal map. Figure 2 illustrates a typical set of texture maps used for a female face: an albedo map, normal map, and two wrinkle maps. The normal map and the wrinkle maps store surface normals in tangent space [Blinn78]. The first wrinkle map encodes wrinkles for a stretched expression (exaggerated surprise expression: eyes wide open, eyebrows up, forehead wrinkled, mouth open) and the second wrinkle map encodes wrinkles for a compressed expression (think of sucking on a sour lemon: eyes squinting, forehead compressed down towards eyebrows, lips puckered, chin compressed and dimpled).

Figure 2. *Ruby's face textures (left to right): albedo map, tangent space normal map, tangent space wrinkle map 1 for stretched face, and tangent space wrinkle map 2 for compressed face.*

As usual, the normal map encodes fine surface detail such as pores, scars, or other facial details. Thus the normal map acts as a base layer that is added to a given wrinkle map. Because the normal map includes important detail, we don't want to simply average the normal map with the wrinkle maps or some of surface details may be lost. Listing 1 includes HLSL shader code for adding a wrinkle map with a normal map in a way that preserves the details of both maps.

```
// Sample the normal map and the wrinkle map (both are in tangent space)
// Scale and bias to get the vectors into the [-1, 1] range
float3 vNormalTS = tex2D( sNormalMapSampler, vUV ) * 2 - 1;
float3 vWrinkleTS = tex2D( sWrinkleMapSampler, vUV ) * 2 – 1;

// Add wrinkle to normal map
float3 vWrinkledNormal = normalize( float3( vWrinkleTS.xy + vNormalTS.xy,
                                            vWrinkleTS.z  * vNormalTS.z ) );
```

Listing 1. *An example HLSL implementation of adding a normal map and a wrinkle map (both are in tangent space). Both maps include important surface details that must be preserved when they are added together.*

4.4 Wrinkle Masks and Weights

In order to have independently controlled wrinkles on our character's face, we must divide her face into multiple regions. Each region is specified by a mask that is stored in a texture map. Because a mask can be stored in a single color channel of a texture, we are able to store up to four masks in a single four channel texture as shown in Figure 3. Using masks allows us to store wrinkles for different parts of the face, such as chin wrinkles and forehead wrinkles, in the same wrinkle map and still maintain independent control over wrinkles on different regions of our character's face.

Each wrinkle mask is paired with an animated wrinkle weight. Wrinkle weights are scalar values and act as influences for blending in wrinkles from the two wrinkle maps. For example, the upper left brow (red channel of left most image in Figure 3) will have its own "Upper Left Brow" weight. Each weight is in the range [-1, 1] and corresponds to the following wrinkle map influences:

- **Weight == -1 :** *Full influence from wrinkle map 1 (surprised face wrinkles)*
- **Weight == 0 :** *No influence from either wrinkle map (just the base normal map)*
- **Weight == 1 :** *Full influence from wrinkle map 2 (puckered face wrinkles)*

A given weight smoothly interpolates between the two wrinkle maps; at either end of the range one of the wrinkle maps is at its full influence and at the center of the range (when the weight is zero) neither of the wrinkle maps has an influence on the underlying normal map. Listing 2 gives HLSL shader code that implements this weighting and blending scheme.

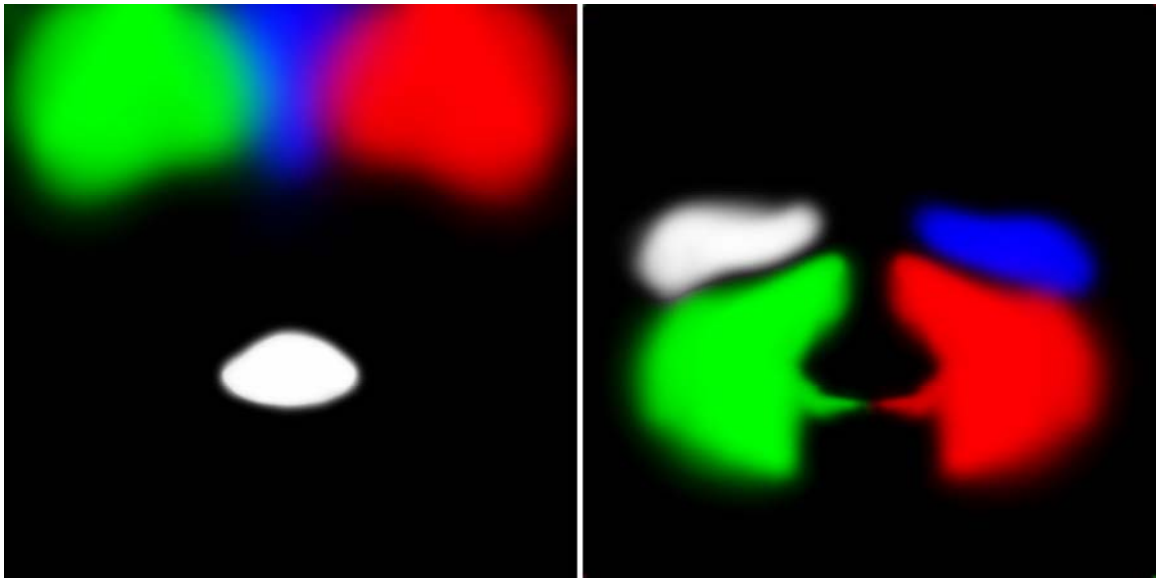

Figure 3. *Eight wrinkle masks distributed across the color and alpha channels of two textures (white represents the contents of the alpha channel). [Left] Masks for the left brow (red), right brow (green), middle brow (blue) and the lips (alpha). [Right] Masks for the left cheek (red), right cheek (green), upper left cheek (blue), and upper right cheek (alpha). We also use a chin mask which is not shown here.*

4.5 Conclusion

An efficient technique for achieving animated wrinkle maps has been presented. This technique uses two wrinkle maps (corresponding to squished and stretched expressions), a normal map, wrinkle masks, and artist animated wrinkle weights to independently control wrinkles on different regions of a facial mesh. When combined with traditional facial animation techniques such as matrix palate skinning and morphing this wrinkle technique can produce very compelling results that enable your characters to be more expressive.

4.6 References

[BLINN78] BLINN, J. Simulation of Wrinkled Surfaces, In Proceedings of ACM SIGGRAPH 1978, Vol. 12, No. 3, pp. 286-292, August 1978.

```
sampler2D sWrinkleMask1; // <LBrow, RBrow, MidBrow, Lips>
sampler2D sWrinkleMask2; // <LeftCheek, RightCheek, UpLeftCheek, UpRightCheek>

sampler2D sWrinkleMapSampler1; // Stretch wrinkles map sampler
sampler2D sWrinkleMapSampler2; // Compress wrinkles map sampler
sampler2D sNormalMapSampler;   // Normal map sampler

float4 vWrinkleMaskWeights1; // <LeftBrow, RightBrow, MidBrow, Lips>
float4 vWrinkleMaskWeights2; // <LCheek, RCheek, UpLeftCheek, UpRightCheek>

// Compute tangent space wrinkled normal
float4 ComputeWrinkledNormal ( float2 vUV )
{
   // Sample the mask textures
   float4 vMask1 = tex2D( sWrinkleMask1, vUV );
   float4 vMask2 = tex2D( sWrinkleMask2, vUV );

   // Mask the weights to get each wrinkle map's influence
   float fInfluence1 = dot(vMask1, max(0, -vWrinkleMaskWeights1)) +
                       dot(vMask2, max(0, -vWrinkleMaskWeights2));

   float fInfluence2 = dot(vMask1, max(0, vWrinkleMaskWeights1)) +
                       dot(vMask2, max(0, vWrinkleMaskWeights2));

   // Clamp the influence [0,1].  This is only necessary if
   // there are overlapping mask regions.
   fInfluence1 = min(fInfluence1, 1);
   fInfluence2 = min(fInfluence2, 1);

   // Sample the normal & wrinkle maps (we could branch here
   // if both influences are zero).  Scale and bias to get
   // vectors into [-1, 1] range.
   float3 vNormalTS = tex2D(sNormalMapSampler, vUV)*2-1;  // Normal map
   float3 vWrink1TS = tex2D(sWrinkleMapSampler1, vUV)*2-1;// Wrinkle map 1
   float3 vWrink2TS = tex2D(sWrinkleMapSampler2, vUV)*2-1;// Wrinkle map 2

   // Composite the weighted wrinkle maps to get a final wrinkle
   float3 vWrinkleTS;
   vWrinkleTS = lerp( float3(0,0,1), vWrink1TS, fInfluence1 );
   vWrinkleTS = lerp( vWrinkleTS, vWrink2TS, fInfluence2 );

   // Add final wrinkle to the base normal map
   vNormalTS = normalize( float3( vWrinkleTS.xy + vNormalTS.xy,
                                  vNormalTS.z * vWrinkleTS.z ) );

   return vNormalTS;
}
```

Listing 2. *An example HLSL function that takes a UV texture coordinate as an argument and returns a wrinkled normal that may be used for shading.*