Real-time Simulation of Wrinkles

Clausius Duque G. Reis, José Mario De Martino, Harlen Costa Batagelo
Department of Computer Engineering and Industrial Automation
School of Electrical and Computer Engineering
State University of Campinas, CP 6101
13083-970, Campinas, SP, Brazil
{clausius, martino, harlen}@dca.fee.unicamp.br

ABSTRACT

Skin wrinkles add realism and expressiveness to 3D facial animation. Modeling and animation of facial wrinkles have been challenging tasks due to the variety of conformations and details subtleness that wrinkles can exhibit. In this paper, we describe a method for real-time simulation of wrinkles taking advantage of the processing power of current GPUs. Our approach is based on a GPU shader program that uses a simple normal mapping approach to apply wrinkles according to influence areas on the face. For efficiency, all the data required for the simulation is comprised in 3D textures. We developed a reusable and extensible XNA component that implements the shader and tools for generating the necessary 3D textures. Currently, this component uses pre-computed facial expressions, which are linear interpolated on the CPU. Nevertheless, it can be easily extended to other deformation approaches, such as key-frames or virtual muscles, without increasing the amount of texture data. We present a description of the developed component, applications and screenshots to illustrate the results.

Keywords

Shader, HLSL, Texture arrays, Areas of influence, Wrinkles, Facial animation, XNA.

1. INTRODUCTION

The display of visual details in virtual models can be divided in two distinct approaches: Geometry based and texture based, but it is possible to have hybrid solutions. With perturbations directly into the geometry it is possible to achieve high realistic results; however this level of details is directly proportional to the necessary number of vertices and polygons in the model. On the other side, the use of texture details does not require a high detailed model; we apply the real objects details directly on the 3D model, but these objects realism will not be convincing.

Techniques such as bump, normal, parallax and relief mapping, among others, allow us to considerably enhance the final results without the need to modify the model topology. Using normal maps, which replace the normal from each pixel, we can create the impression that we are using high polygon models when, in fact, we are using medium or low polygon models.

Physics based solutions normally need to compute large amounts of data. This approach tends to delay the presentation of the results, reducing the system overall speed and preventing their use in real-time.

With the advancement of the graphics cards processing capacity, several areas are using this computational power to accelerate the applications speed. In areas such as game development, movies and animations the use of shaders has become very common, due to their ability to produce good graphics results with little processing.

The method presented in this article demonstrates how to use today's GPU to simulate the appearance of skin wrinkles in real time using texture arrays, areas of influence and the new capabilities from current graphical boards.

On the following sections, it will be described the techniques used, the generation of the necessary information, the use of that information on the shader side, the reuse of the developed component on different approaches and graphical results.

2. RELATED WORK

The great majority of the works developed on computer graphics wrinkles is directed to the facial animation area.

Bando et al. [BKN02] presents a hybrid solution to generate fine and large scale wrinkles. For that, the author uses hierarchical Voronoi division to create

the fine scale wrinkles, and cubic Bezier curves defined by the user to produce the large ones. He presents the small wrinkles using bump mapping [BLI78] and the large wrinkles were set directly onto the models mesh by adaptive refinement [VOT99] them and changing the vertices position with displacement map. The technique produced results in an interactive frame rate of approximately 5.0 to 6.5 fps and it took a necessary initial time of 10 seconds to generate the fine-scale wrinkles. The performance results were taken on a XEON 2GHz with a Wildcat II 5110 graphics card, producing an image and texture with sizes of 512x512.

Boussiex et al. [BKTK00] presented an image based method where they used generic masks to simulate the aging of permanent wrinkles. They also presented a method to simulate the elastic behavior of the skin due to the aging process. They used layers with different materials and a finite element method for computing the deformation of the skin aging.

Mikael Nordenberg [NOR03] on the other hand, used a bump mapping shader to present wrinkles on a virtual model when its forehead was compressed. His proposal included pre-computation calculations to store the direction of the wrinkles at each vertex and also per frame calculations, responsible to calculate the model geometry compression and determine if the wrinkles should be displayed. The bump mapping technique was used to present the wrinkles with a static wrinkles map texture.

Pei-Hsuan et al. [TLYLO06] used motion capture to extract in 2D and 3D the facial animations and wrinkles from a video tracking source. The system generated textures and normal maps [KIL00] that were interpolated on key-framing animation. However, the method proposed does not enable the user to choose which facial details will be presented; it just reproduces the captured information.

Other proposals, like Loviscach [LOV06], presented a technique to produce wrinkles in real-time with bump mapping where, instead of physical dynamics, kinematical preservation of length through buckling was used, requiring to send some information to the GPU like vertex position, vertex neighbors and direction of wrinkles to compute the compression of the model's mesh and present the final result. A pre-processing stage could be very lengthy depending on the model's number of vertices.

Wang [WWY06] presented an approach based on a governing curve and influence regions that model the distinct wrinkle shapes to represent the different material properties of non-rigid objects. The results are presented with displacement map and adaptive refinement, what causes the system to lose speed and make the wrinkling operation last around 2 seconds to process on a Pentium 4 1.8GHz with 512 MB RAM for a head model with 66737 polygons.

Other system that generated expression wrinkles direct on the mesh was presented by Li et al. [LBKL07], where they divided the face in wrinkles sub regions and created some wrinkles lines with their own algorithm. Then they used displacement map to interpolate between the regions and present the wrinkles. With this technique, they manage to generate the wrinkles on a 16659 triangles facial mesh in approximately 594ms, using a Pentium Dualcore 2.80GHz and a NVIDIA GeForce 6200 TurboCache graphic card.

In our approach we are not trying to create the wrinkles, but present them in real-time and in an easy way. If the user wants to add or remove wrinkles, it can be done easily since we are working with a dynamic number of areas of influence.

Unlike some authors [BKTK00, BKN02, WWY06, LBKL07], we did not work with vertex displacement or adaptive refinement, instead of that, we chose to use normal maps to present the surface details, making it possible to use low or medium polygon models. It also allowed us to improve our system's FPS speed, since the performances obtained by Bando [BKN02], Wang [WWY06] and Li [LBKL07] were not suitable for real-time facial animation applications. Loviscach [LOV06] managed to get very good performance results during the execution of the system, although the pre-processing stage delay could become a problem in areas like game development.

We also did not use any physical simulation approach as used by Wang [WWY06] and Boussiex [BKTK00] because of the speed bottleneck, instead of that, we preferred to work with pre-made textures and normal mapping shader, which gave us a better visual result than bump mapping approaches [BKN02, NOR03, LOV06].

Our system presents wrinkles on the entire face, whereas the normal map possesses these details in it and not just on the forehead, as presented by Nordemberg [NOR03]. It also enables to choose the area in which the wrinkles should appear on the face using areas of influence, instead of just reproducing the recorded information as Pei-Hsuan [TLYLO06] or morphing between the different aged normal maps like Boussiex [BKTK00].

During the initialization of the system we add a few key textures, such as diffuse texture, normal maps and several grayscale textures with the wrinkles area of influence. After that, texture arrays comprising all this texture information are generated and passed to the shader. These areas of influence are

used to calculate which wrinkles are going to be presented depending of an array with the activation percentage of each area.

3. OUR APPROACH

In this section we present our method for simulating facial wrinkles, describe a tool for packing the required data and present a shader program that uses these data for the simulation of wrinkles in real time.

Our proposal is to work areas of influence and normal maps to present wrinkles using a Shader Model 3.0 compliant graphical board. This is necessary because of some new functionality implemented on them that we use on our logic. We provide all the necessary information to the system comprised in 3D textures, but we do not use them as volume textures. Instead of that, we work with the 3D textures as texture arrays containing all the necessary data in separated layers which we use to loop and calculate the wrinkles presentation inside the GPU shader.

Among the information needed, we send a diffuse texture, two normal maps containing pores, scars and other details, but just one of them will contain wrinkles, and several areas of influence. The number of areas of influence will depend on how many wrinkles areas the user wants to control.

With a floating point vector containing the percentage of activation of every area of influence we control whether the wrinkle will be displayed or not.

In the following sections we explain the use of each component of our technique.

3.1 The necessary textures

Diffuse texture: This texture is responsible for the model skin color; it is a texture from a human face stretched on a plane with the region texture coordinates correspondent to the 3D model vertex (Figure 1). In our case we create the model using MAYA [MAY07].

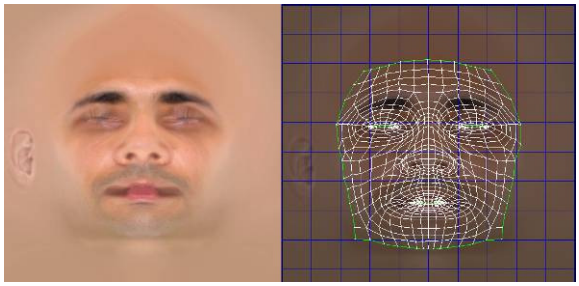

Figure 1. Diffuse texture (left) and texture coordinates in face texture (right)

Normal maps: The system needs two normal maps [KIL00] that along with the diffuse texture will result in the color for each pixel of the rendered image. These normal map textures could be generated by some software that uses a high polygon model to export all the details modeled onto the geometry [ATI07]. Other way is to use height maps and then use some software to convert them to normal maps [CRB07, NVI07].

The first normal map (Figure 2: Left) has no details or perturbations on the pixel's normal; the second one (Figure 2: Right) has details, or perturbations, for all the possible wrinkles on the model's face when that is deformed. Notice that we are not trying to physically simulate the human skin behavior, but present wrinkles in real time on the existent models with the normal mapping technique and the use of texture array.

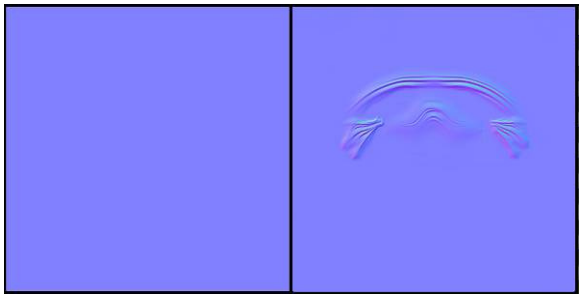

Figure 2. Normal map without wrinkles (left) and normal map with forehead, nose and crown's feet wrinkles (right).

Areas of influence: They are basically grayscale textures that have the wrinkles visibility information for each region of the face model, in other words, they are textures that inform the shader the places where each wrinkle should or should not appear. These areas can be compared to the transparency information within the RGBA images. For each area of influence, the only visible region will be coincident to the wanted wrinkles; all the remaining area will be 100% transparent. The user will paint the region of the texture with the percentage they want to display the wrinkles. If the pixel is white the system will not present wrinkles, but if the pixel is black the wrinkles will be 100% visible. Every value between white and black will gradually present the wrinkles. Our implementation demands one area of influence texture for each wrinkles region that the user wants to have control. These textures can be generated in any image editing software.

In Figure 3 we illustrate two areas of influence, the left and right forehead wrinkles. As seen on the top left part of Figure 3, the left forehead overlaps the area where the wrinkles will be displayed and the top right shows the final area of influence generated by the user.

We could have as many areas of influence as the computer and graphic board allow us, but the more

 ISBN 978-80-86943-16-9

areas we have, the longer it will be necessary to calculate each pixel.

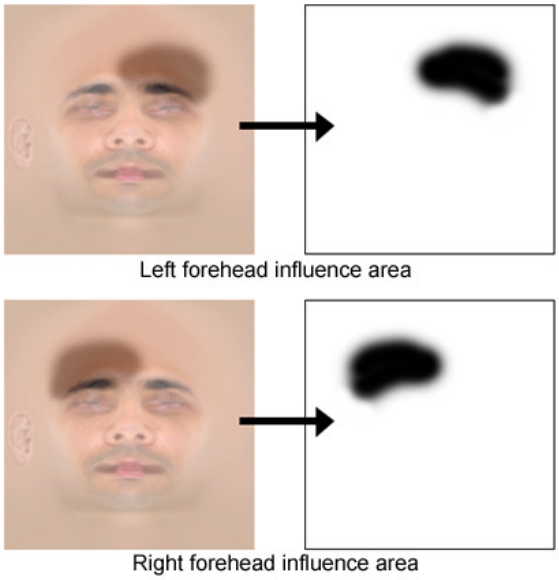

Figure 3. Wrinkles area of influence.

On figure 4 we present a proposal for a basic setup of the areas of influence containing 8 distinct areas. These areas could be changed in any necessary way to produce the user desired results.

We chose this basic setup based on the face subdivision areas proposed by Li [LBKL07]. In our tests these areas have presented good visual results and FPS performances.

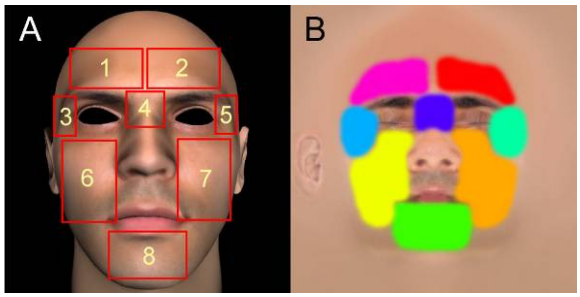

Figure 4. Proposal to areas of influence setup.

3.2 Generating the texture arrays

All the necessary textures must be added in the system, since they will compose the final texture arrays that will be sent to the shader. The first one is a high resolution 512x512x3 color texture array composed by the diffuse texture and the normal maps, and the second one is a low resolution 128x128xN grayscale texture array with the N areas of influence information. This process is carried out only one time during all the system.

This step could be eliminated adding layered textures pre-constructed in some specific software [VTT07] for this end. However this process is generally a laborious task, and for this reason we add to the system the capacity to generate these textures, once all the information is available.

The basic project of the texture arrays for the functioning of the system is presented in figure 5.

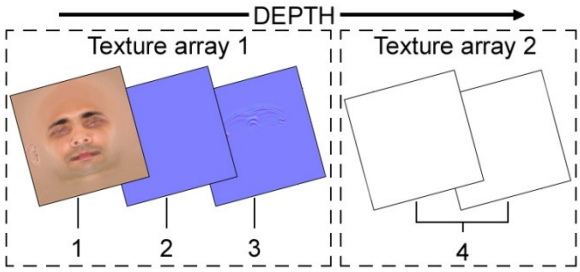

Figure 5. Basic setup of the texture arrays.

Where:

1: Diffuse texture.

2: Normal map without wrinkles.

3: Normal map with wrinkles.

4: Areas of influence.

Depending on how many regions the user desires to control, we can have a different number of influence areas.

To produce the texture arrays, we use an algorithm (Figure 6) where we draw each pixel of the image copying the corresponding information of the regular 2D textures pixel:

```
1   Color[] myBits =
2       new Color[Width * Height * Depth];
3
4   for (int d = 0; d < Depth; d++)
5   {
6       Color[] bits2D = new Color[Width * Height];
7       textures2D[d].GetData<Color>(bits2D);
8
9       for (int x = 0; x < Width; x++)
10      {
11          for (int z = 0; z < Height; z++)
12          {
13              myBits[x + z * Width + d * Width * Height] =
14                  bits2D[x + z * Width];
15          }
16      }
17  }
18
19  texture3D.SetData<Color>(myBits);
```

Figure 6. Texture arrays generation algorithm.

On this C# piece of code we show how the XNA manages to recover the bits of the texture directly in RGBA format, as can be seen at figure 6, lines 6 and 7, what facilitates the work sufficiently to manipulate the image pixels.

For the texture array composed by the diffuse texture and the normal maps, we create an array of Colors in format RGBA (Line 1) and set its size to match the texture array. For each depth loop (Line 4), we read the information of the 2D texture that we want to insert in the texture array (Line 6 and 7). After that we loop through the Width and Height of

the textures inserting them in the array (Lines 9 to 17). Finally we have the texture array bits to return to the system (Line 19).

The same algorithm is used to generate the texture array composed by the areas of influence, but instead of using a RGBA texture array, we create a grayscale texture array to save GPU memory, since the areas of influence are grayscale textures and we do not use the green, blue and alpha channels.

We have used 3D textures instead of several 2D textures because with 2D textures, every time we add or remove an area of influence we would need to change the shader to manage the new item, which is not necessary with 3D textures since we can do it dynamically, performing loops between the depths of the texture, a necessary attribute to our technique that could not be done with regular 2D textures.

3.3 Passing the information

Having in hands the texture array we contend all the necessary information for the calculation of normal mapping and the exhibition of wrinkles, we can start to pass the information to the GPU.

Beyond the necessary basic information for visualization (world, view and projection matrices) and the texture arrays generated by the system, we need to inform to the shader the activation percentage of each wrinkle on determined influence area. For this, we need to create a vector of equal size to the depth of the texture array with the areas of influence. If the texture has 4 areas of influence, our vector will have a length equals 4 and each position of the vector receives the activation percentage of its corresponding texture.

In our approach we use simple interpolation between some pre-modeled facial expressions, and we define percentages of activation of the areas of influence for each model. Other techniques however could be used in this system. A good example would be the use of virtual muscles, where the areas of influence textures would be generated in accordance with the vertices influenced directly by the muscle, and the percentage of activation of each one of these areas would be given by the force of activation of the proper muscle.

As it can be seen, the implementation can be different, but the passed information will always be the same one.

3.4 The shader

The used technique is a modification of the normal mapping shader. The difference is that our implementation calculates the color result of two normal maps and interpolates between them using the areas of influence activation for the final pixel color, displaying wrinkles in the surface of the model.

The interpolation is calculated in the GPU after the calculation of the two normal mapping final colors by the algorithm on figure 7.

```
1   ...Compute the first normal mapping (finalColor1)...
2
3   ...Compute the second normal mapping (finalColor2)...
4
5   finalColor2.a = 0.0;
6
7   for (int i = 0; i < Depth; i++)
8   {
9      float4 pixelAlpha =
10        tex3D(
11          AreasOfInfluenceSampler,
12          float3(
13            input.texCoord.x,
14            input.texCoord.y,
15            ((1.0 / Depth) * i) + 0.01
16          )
17        );
18
19     finalColor2.a =
20        finalColor2.a + (pixelAlpha.x * Influency[i]);
21  };
22
23  float4 finalColor =
24     lerp(finalColor1, finalColor2, finalColor2.a);
25
26  return finalColor;
```

Figure 7. Internal shader loop to compute wrinkles exhibition based on areas of influence.

After computing the pixel resultant color based on two normal mapping textures (Lines 1 and 3), we clear the alpha information of the second final color (Line 5), and loop through all areas of influence of the texture array (Lines 7 to 21) adding the corresponding pixel values in the areas of influence multiplied by the percentage of activation of each area in the alpha field (Lines 19 and 20), resulting in a normal mapping texture with the alpha channel value equals the sum of all influence areas.

In line 20 we use only the red channel of the pixel because the user adds grayscale textures as areas of influence having only one channel. However, the texture array having the diffuse texture and normal maps will have 4 channels, red, green, blue and alpha, since it is created in a RGBA format.

Once we have the final color from both normal maps and the wrinkles activation in the alpha channel (Line 20), we can interpolate between the final colors (Line 23 and 24), returning only the desired wrinkles to be displayed.

We need to use graphical boards compliant with Shader Model 3.0 because of some limitations from the previous models. On line 7 we use the "Depth" variable to set the end of the loop, which could not be done with Shader Model 2.0. This is necessary because we want to create a generic shader when dealing with different number of areas of influence. If we had not done that, every time we inserted or removed an area of influence we would have to change the shader algorithm to match the depth of the texture array.

Our system uses an array to send the areas activation percentage, which on Shader Model 3.0 is limited to a length of 19. If the user wants to work with more than 19 areas of influence, this limit could be surpassed using an image with one dimension. For example: If we wanted to send 30 areas of influence we would need to create an image with width equals 30 and height equals 1 and implement on the shader a reader like it is done with the depth on line 15 of figure 7.

4. RESULTS

To demonstrate the use of our technique, we used a 3D model with 5960 vertices and interpolated it between several expressions, calculating the vertex normal, tangent and binormal at each frame. The solution was implemented with the Visual C# Express [CEX07] and the Microsoft's XNA framework [XNA07]. As a result, we manage to get high frame rates in different computers and graphical boards. On tables 1 and 2 we compare the performance results between a notebook and a desktop computer to render the wrinkles, both with 256 MB RAM and projected area of 800x600 pixels.

Notebook Compaq Presario V6210BR AMD SEMPROM with ~1.8GHz NVIDIA GeForce 6150 Go 256 MB (Shared)	
FPS	55 FPS
Frame time	19,6078 (ms)

Table 1. Notebook computer performance results.

Desktop computer AMD SEMPROM with ~1.5GHz NVIDIA GeForce 7600 GS 256 MB (Dedicated)	
FPS	42 FPS
Frame time	25,4803 (ms)

Table 2. Desktop computer performance results.

We get the performance results show in table 1 and table 2 on tests performed to render all model animations, independently if we were working with 2 or 16 areas of influence (Figure 8 to 10). It's important to note that these are average values and depending of the system overload they could also be lower or even higher.

Figure 8. Neutral face (Left) and surprised face with 2 areas of influence activated (right), left and right forehead wrinkles.

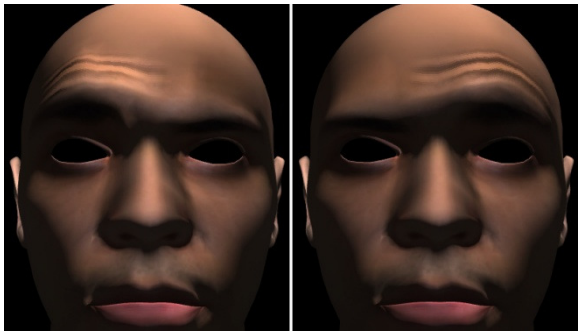

Figure 9. Left and right eyebrow.

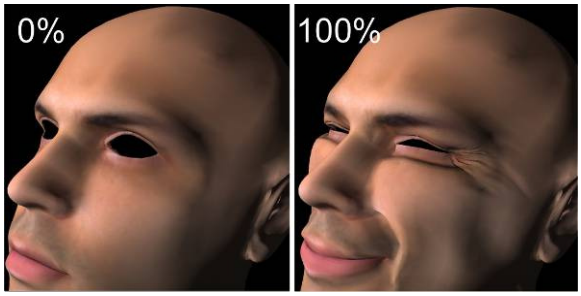

Figure 10. Left crow's feet.

To produce the results of figure 8 we used 2 areas of influence. The same areas were used to produce the results on figure 9, but they were used once at a time. Figure 10 uses one area of influence.

5. CONCLUSION

In this paper we have presented our approach for the successfully display of wrinkles on a 3D model surface. We demonstrated how to use today's GPU to exhibit wrinkles in real-time using texture arrays, areas of influence and current graphical boards. We managed to get satisfactory performance results on two modest computers with low memory and limited hardware.

The use of texture arrays facilitates the process of changing the facial wrinkles setup or even to switch between several implementations such as virtual muscles or key frames, once the same information is passed.

We produced results in a virtual face model, but our proposal allows its use for other purposes like wrinkles on the hands, neck and fabric, among others.

The system allows overlap of areas of influence, which means that we can draw areas of influence that control the wrinkles of other areas. This is possible since we are using textures to show the wrinkles, and not to calculate them.

For future work, we intend to change from the shader effect to one that produces wrinkles with silhouettes. The reason for this is that the normal mapping shader produces good results, but if we look

closer to the model we want to show some relief instead of a flat surface.

6. REFERENCES

[ATI07] ATI's normal map generator program, http://www2.ati.com/developer/NormalMapper-3_2_2.zip (Last access: 10/25/2007)

[BKN02] Bando, Y.; Kuratate, T.; Nishita, T., A simple method for modeling wrinkles on human skin. Proceedings of 10th Pacific Conference on Computer Graphics and Applications 9-11 Oct. 2002, 166-175, 2002.

[BKTK00] Boissieux, L.; Kiss, G.; Thalmann, N. M.; Kalra, P., Simulation of Skin Aging and Wrinkles with Cosmetics Insight. Proceedings of Eurographics Workshop on Computer Animation and Simulation 2000, 15-27, 2000.

[BLI78] Blinn, J., Simulation of Wrinkled Surface. Proceedings of SIGGRAPH 78, 286-292, 1978.

[CEX07] Visual C# Express, Microsoft., http://msdn.microsoft.com/vstudio/express/visualcsharp/download/ (Last access: 10/17/2007).

[CRB07] Crazybump, http://www.crazybump.com/ (Last access: 10/24/2007)

[KIL00] Kilgard, M., A practical and robust bump-mapping technique for today's GPU's. Technical report, NVIDIA Corporation, February, 2000.

[LBKL07] Li, M.; Yin, B; Kong, D; Luo, X., Modeling Expressive Wrinkles of Face For Animation. Fourth International Conference on Image and Graphics. 874-879, 2007.

[LOV06] Loviscach, J., Wrinkling Coarse Meshes on the GPU. Proceedings of Eurographics, Computer Graphics Forum 25(3), 467-476, 2006.

[MAY07] Autodesk Maya, http://www.autodesk.com (Last access: 10/25/2007)

[NOR03] Nordenberg, M., Modelling and rendering dynamic wrinkles in a virtual face. TMH/KTH MSc thesis, 2003.

[NVI07] Nvidia's DDS plugin for Photoshop, http://developer.nvidia.com/object/nv_texture_tools.html (Last access: 10/24/2007)

[TLYLO06] Tu, P; Lin, I; Yeh, J; Liang, R; Ouhyoung, M., Expression Detail Mapping for Realistic Facial Animation. Proceedings of CAD/Graphics 2003, 20-25, 2003.

[VOT99] Volino, P.; Thalmann, N. M., Fast Geometrical Wrinkles on Animated Surfaces. Proceedings of WSCG '99, 1999.

[VTT07] Volume texture tool, http://www.mdxinfo.com/resources/volumetexturetool.php (Last access: 10/25/2007)

[WWY06] Wang, Y; Wang, C. C. L.; Yuen, M. M., Fast energy-based surface wrinkle modeling. Computers & Graphics 30(1), February 2006: 111-125, 2006.

[XNA07] XNA Game Studio Express, Microsoft, http://creators.xna.com/Resources/Essentials.aspx (Last access: 10/17/2007).