Noname manuscript No.
(will be inserted by the editor)

Curling and Clumping Fur Represented by Texture Layers

Paulo Silva · Yosuke Bando · Bing-Yu Chen · Tomoyuki Nishita

Received: date / Accepted: date

Abstract Fur is present in most mammals which are common characters in both movies and video-games, and it is important to model and render fur both realistically and quickly. When the objective is real-time performance, fur is usually represented by texture layers (or 3D textures), which limits the dynamic characteristics of fur when compared with methods that use an explicit representation for each fur strand.

This paper proposes a method for animating and shaping fur in real-time, adding curling and clumping effects to the existing real-time fur rendering methods on the GPU. Besides fur bending using a mass-spring strand model embedded in the fur texture, we add small scale displacements to layers to represent curls which are suitable for vertex shader implementation, and we also use a fragment shader to compute intra-layer offsets to create fur clumps. With our method, it becomes easy to dynamically add and remove fur curls and clumps, as can be seen in real fur as a result of fur getting wet and drying up.

Keywords Fur Textures · Fur Curling · Fur Clumping · Real-Time Rendering

CR Subject Classification Computer Graphics: I.3.7 - Three-Dimensional Graphics and Realism - Animation

Paulo Silva · Tomoyuki Nishita
The University of Tokyo
E-mail: {paulo,nis}@nis-lab.is.s.u-tokyo.ac.jp

Yosuke Bando
Toshiba Corporation and The University of Tokyo
E-mail: yosuke1.bando@toshiba.co.jp

Bing-Yu Chen
National Taiwan University E-mail: robin@ntu.edu.tw

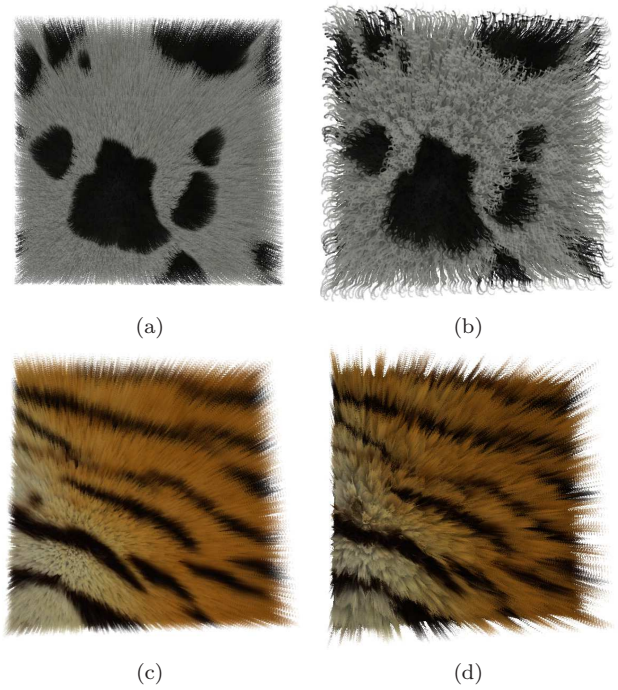


Fig. 1 Examples of real-time dynamic fur manipulations possible with our method: (a) normal straight, (b) curly (c) dry, and (d) clumped wet furs.

1 Introduction

In real life, furry surfaces tend to be fluffy and soft, and fur often curls and forms clumps (i.e., fur strands gather into a cluster) when wet. In computer graphics, existing fur related researches concentrate either on realism or real-time rendering. In the latter, fur is usually represented by static 3D textures or texture layers, which limits the dynamic characteristics of fur when compared with methods that use an explicit representation for each fur strand.

In this paper we propose a method for manipulating the fur shape while maintaining real-time performance. We focus on (un)curling and (un)clumping effects visible when fur becomes wet or dries up, and add these effects to the existing real-time fur rendering methods on GPU [Lengyel et al., 2001]. We generate a 2D texture mask representing wet fur clumping areas, so that portions of the object surface can be selectively subject to wetness. We position a mass-spring fur strand model over the object surface, and use it to displace the fur texture layers to represent large scale deformation of the fur. This algorithm is aimed to be implemented in the GPU vertex shader.

We also use the same wetness mask to know if a strand is in a clump region, and if so we compute the displacement that the strand should suffer due to the clumping effect. This algorithm is suitable to be implemented in the GPU fragment shader, because different displacements need to be applied to each point within a layer, and we realize this using texture coordinate manipulation.

As a result, our method can dynamically add and remove fur curls and clumps, as can be seen in real fur when getting wet and drying up. Examples illustrating the proposed method are shown in Fig. 1. To the best of our knowledge, the effects possible with our method were not seen performed in real-time in the previous research.

2 Related Work

In literatures, there can be found several kinds of approaches to render fur or fur-like geometric detail (e.g. grass). The main differences between these methods are the way the fur data is stored and the way the rendering algorithm produces the final image. There are three main approaches: methods that store the fur data using textures, methods that use lines or curves to represent each fur strand individually, and methods that generate fur procedurally using a mathematical model for its structure. Researches representative of each of these methods are summarized as follows.

2.1 Fur Represented by Textures

Fur rendering using textures for its representation was first presented by [Kajiya and Kay, 1989]. The authors store fur illumination parameters in the texture and ray-trace it to render the final image. Later [Neyret, 1998, Meyer and Neyret, 1998] proposed a method to represent a volume by using several semi-transparent slices in planes parallel to the xy, yz and zx planes. These slices are stored into three separate 3D textures. This method became popular for real-time fur rendering through the work of [Lengyel et al., 2001], in which the authors used one 3D texture and a single extra layer to represent fur. This method served as a basis to several similar techniques [Isidoro and Mitchell, 2002, Bakay et al., 2002, Papaioannou, 2002, McGuire and Hughes, 2004, Habel et al., 2007, Tariq and Bavoil, 2008] most using the power of the ongoing GPU development to improve a single part of the rendering process.

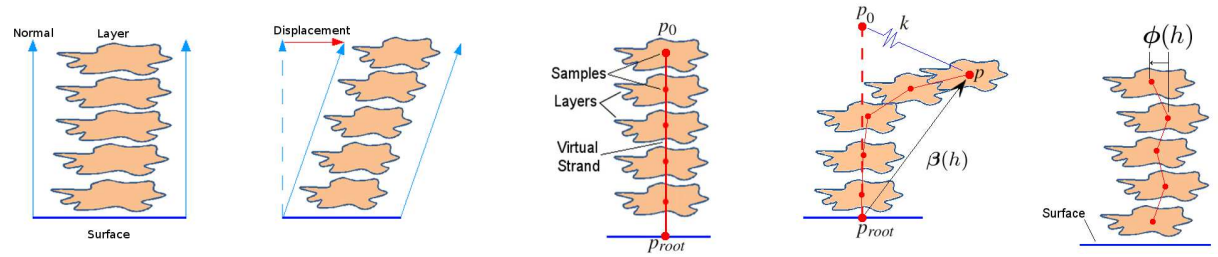
A method for rendering fur using precomputed radiance transfer was presented by [Kloetzli, 2006]. [Banisch and Wüthrich, 2006] presents a throughout discussion on mass-spring simulation for the animation of fur and grass represented by texture layers. More recently, research targeting the view dependency of the number of layers on the perception of the fur detail, and fur shadowing were presented by [Yang et al., 2006, 2008, Sheng et al., 2009]. A method for mapping fur geometry directly onto the mesh was presented by [Elber, 2005, Jeschke et al., 2007]. A method for painting fur geometry directly on the model was proposed by [Owada et al., 2008]. Finally, [Jiao and Wu, 2009] presented a method for simulating weathered fur. However, dynamic curling and clumping effects cannot be found in none of the above literature.

2.2 Fur Represented by Geometric Primitives

Classic examples of this approach were presented by [Csuri et al., 1979, Miller, 1988, Gelder and Wilhelms, 1997]. A method for rendering wet fur clumping was introduced by [Bruderlin, 2000, 2003]. This method concentrates primarily on representing the shape the fur takes when wet, but is not real-time oriented. We take a similar approach, but we target real-time rendering. Some optimizations to [Bruderlin, 2000, 2003] were proposed by [Takeuchi et al., 2009], but real-time performance was not achieved.

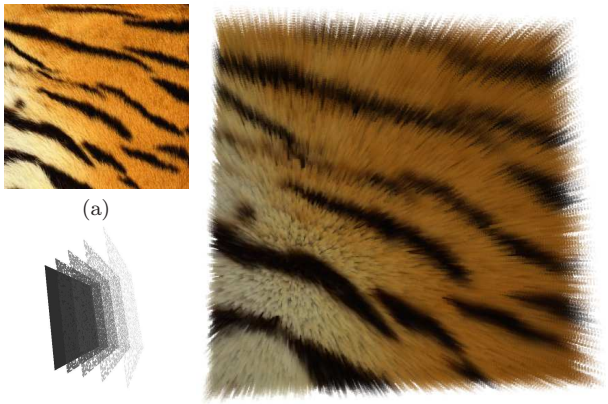
2.3 Procedural Fur

A classic approach in this area is to use a probabilistic model as in the work by [Perlin and Hoffert, 1989, Goldman, 1997]. A particle based cell generator was presented by [Fleischer et al., 1995]. Finally, [Kowalski et al., 1999] proposed a method which generates procedural fur-like boundaries, targeting non-photorealistic applications.



(a) (b) (c) (d) (e)

Fig. 2 (a) An example of previous methods [Lengyel et al., 2001, Tariq and Bavoil, 2008] for repenting straight fur. (b) The fur with a tangent space displacement (red arrow) applied to the vertex normal. (c) Our method uses a virtual strand based on a mass-spring system, which is used to control the fur texture deformation and (d) with external forces applied. (e) Fur curling due to factor $\phi(h)$ represented schematically.



(a) (b) (c)

Fig. 3 (a) Input color texture pattern. (b) Fur geometry texture layers. (c) The result of applying (a) to (b).

(a) (b)

Fig. 4 (a) Fur under the influence of a global force. The white line in the middle of the furry triangle represents the mass-spring strand model presented in Sec. 3.2. The underlying fur texture layers perpendicular to the surface triangle are repositioned during rendering as a result of mass-spring strand model shape changes. (b) A square covered with fur under the influence of wind (red arrow).

3 Curling and Clumping Fur

Our algorithm is oriented towards a GPU implementation and is composed of two phases. The first phase occurs in the vertex shader where we implement a curling or uncurling fur effect. The second phase occurs in the fragment shader where we implement the clumping and rendering.

In our method, the fur representation is based on Lengyel et al.'s [Lengyel et al., 2001] work, extended to support dynamic curling. This rendering algorithm produces the results shown in Fig. 3 (c). Our fur clumping method conceptually resembles Bruderlin's [Bruderlin, 2000] work, but is oriented towards a GPU implementation for real-time rendering.

Our method uses as input a mesh and a texture representing the fur color. We create a wetness mask texture (Fig. 5), which contains the clump regions information. Here we assume a parameterization from the model to the texture is available. In the wetness mask we store the center of the clump, its radius, and its wetness or clump-percent (see Sec. 3.1).

During rendering, in the vertex shader we displace the base surface as explained in Sec. 3.2.2, then in the fragment shader we apply a fur clumping process as described in Sec. 3.3.

3.1 Wetness Mask Generation

In our method each drop of water (or droplet) creates a fur clump and removes curliness of the fur. When droplets overlap, the overlapping areas create a new clump. To store this information we create an additional texture which we call *wetness mask* (Fig. 5). Here we assume a parametrization from the model to the texture is available. The clumps are added by blending droplets into the wetness mask texture. Consider that a droplet hit the input mesh at position c_{mesh}. Let the droplet position $c_{\text{drop}} = (u, v)$ be a texture coordinate in the wetness mask corresponding to c_{mesh} on the object surface. Let r_{drop} be the radius, and $\rho_{\text{drop}} \in [0, 1]$ be the wetness of that droplet, respectively. The droplet position c_{drop} is written to texture channels (R, G) and $(r_{\text{drop}}, \rho_{\text{drop}})$ are written to texture channels (B, A). Let

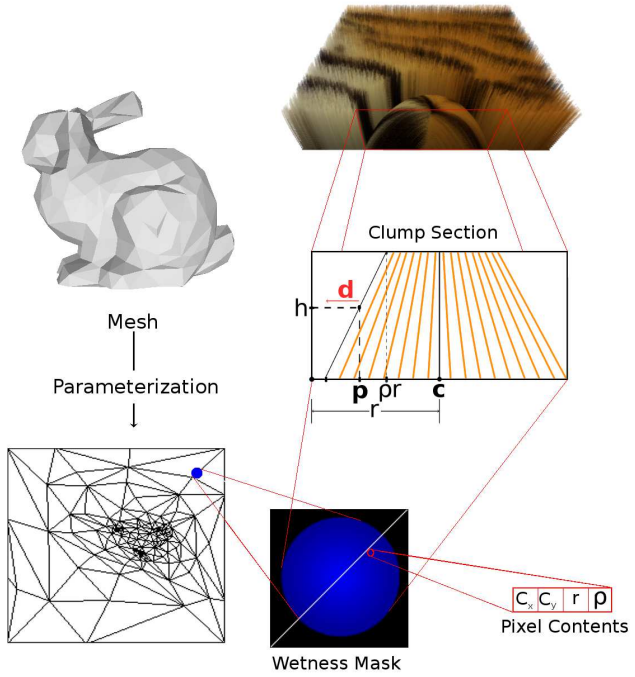
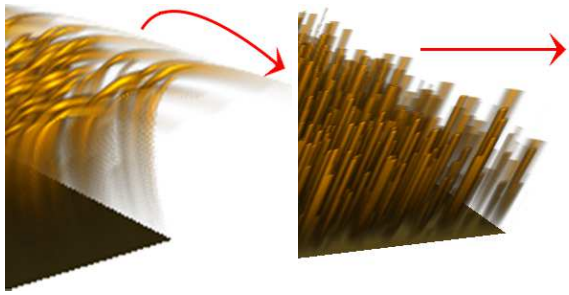


Fig. 5 Using the mesh parametrization and the wetness mask, we compute the texture coordinate displacement **d**. The black area in the wetness mask represents dry fur. The colored area represents the wet areas (see Sec. 3.3). In the clump section we can see: **d** is the fur displacement vector, **p** is the current pixel texture coordinates in the wetness mask space, **c** is the center of the clump, h is the normalized fur length, r is the radius of the clump, and ρ is the wetness or clumping percentage. The clump section illustrates schematically a clump section of the fur in Fig. 9 (a), where in orange we have the strands already displaced and in black (over point **c**) we have the clump center or master-strand as in [Bruderlin, 2000].

c_{mask}, r_{mask} and ρ_{mask} be the values previously written to the wetness texture. A droplet is blended into the wetness mask texture using Alg. 1.

Algorithm 1 Blend Droplet Into Wetness Mask

```
if ‖c_mask − c_drop‖ ≤ r_drop then
  if ρ_mask ≠ 0 then
    ρ_total = clamp(ρ_mask + ρ_drop, 0, 1)
    if ρ_total ≠ 0 then
      α = ρ_drop / ρ_total
      c_mask = (1 − α)c_mask + α c_drop
      r_mask = (1 − α)r_mask + α r_drop
    end if
    ρ_mask = ρ_total
  end if
else
  c_mask = c_drop
  ρ_mask = ρ_drop
  r_mask = r_drop
end if
```



(a) (b)

Fig. 6 Difference in fur shape and highlights when (a) rendered with factor $\boldsymbol{\beta}(h) \neq 0$, and (b) using only a linear tangent space displacement (as in e.g. [Lengyel et al., 2001]). The red arrows in the figures represent the direction in which the layers are displaced.

3.2 Dynamics and Curliness Control

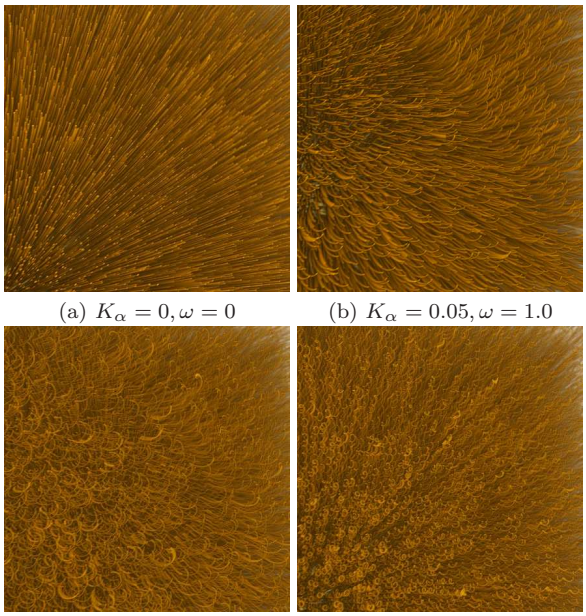
In this section we define our mass-spring model and explain how it is used to control the fur texture layers. Previous methods [Lengyel et al., 2001, Isidoro and Mitchell, 2002, Tariq and Bavoil, 2008, Yang et al., 2006, 2008, Sheng et al., 2009] extrude the surface in the vertex normal direction, while blending successive texture layers on top of each other (see Fig. 2 (a) and (b)). Our method uses an arbitrary displacement function composed by two factors as:

$$\mathbf{v}_{\text{offset}}(h) = \boldsymbol{\beta}(h) + \boldsymbol{\phi}(h). \quad (1)$$

The parameter $h \in [0, 1]$ is the normalized fur length. The factor $\boldsymbol{\beta}(h)$ is responsible for the fur bending due to external forces applied to a mass-spring simulation (see Sec. 3.2.1). The factor $\boldsymbol{\phi}(h)$ controls the fur shape at a finer scale (see Sec. 3.2.2), which we use to control the fur curliness. The factor $\mathbf{v}_{\text{offset}}(h)$ is used in the vertex shader to displace the incoming vertices from the input triangular mesh. These factors are explained in more detail in the next subsections.

3.2.1 Fur Strand Model

In our method we embed a mass-spring strand model per mesh triangle (see Fig. 2 (d)). There are examples of mass-spring systems used for texture space deformation as in [Neyret, 1998], but our model aims at simplicity of implementation and performance. We use only one mass and spring per simulated strand. The particle has initial position $\mathbf{p}_0$, instant position $\mathbf{p}$ and position on the surface $\mathbf{p}_{\text{root}}$. Each fur strand is composed by an initial length $L_0 = \|\mathbf{p}_0 - \mathbf{p}_{\text{root}}\|$, a growth direction normal to the surface, and a single particle with mass m, positioned at the tip of the strand. The behavior is



(a) $K_\alpha = 0, \omega = 0$ (b) $K_\alpha = 0.05, \omega = 1.0$

(c) $K_\alpha = 0.039, \omega = 2.3$ (d) $K_\alpha = 0.009, \omega = 7.2$

Fig. 7 Fur rendering: (a) straight fur, (b) wavy, (c) with few large curls, and (d) with many small curls.

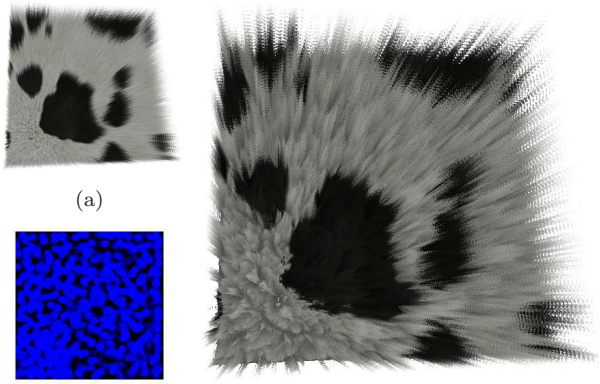
defined simply by

$$\sum_i \mathbf{f}_i = m\ddot{\mathbf{p}}, \tag{2}$$

where $\sum_i \mathbf{f}_i$ is the sum of all the forces on the particle like wind, gravity, etc. The particle position $\mathbf{p}$ is connected to its initial position $\mathbf{p}_0$ by a spring of constant k (see Fig. 2 (d)). The length of the fur strand is loosely constrained by limiting its area of movement to the hemisphere above the surface, represented by the following conditions:

$$\begin{cases} \|\mathbf{p} - \mathbf{p}_{\text{root}}\| & \leq L_0, \\ (\mathbf{p} - \mathbf{p}_{\text{root}}) \cdot \mathbf{n} & \geq 0. \end{cases} \tag{3}$$

The strand shape is approximated by a quadratic Bezier curve defined by $\mathbf{p}_{\text{root}}$, $\mathbf{p}_0$, and $\mathbf{p}$. For N layers, we sample the curve at N points, and these positions are used to displace each layer, as illustrated by the red points in Fig. 2 (c), (d) and (e). In Fig. 4 we can see a rendering illustrating the mass-spring strand (in white) controlling the shape of the surrounding fur, which is represented by texture layers. The rendering comparison using simple tangent displacement and our factor $\boldsymbol{\beta}(h)$ is shown in Fig. 6. This virtual strand allows the fur texture to bend more freely. Note that although fur layers can take an arbitrary position, they are always parallel to the base surface.



(a)

(b) (c)

Fig. 8 (a) Input fur texture. (b) Wetness mask with wet areas displayed in blue. (c) The result of applying to input fur texture (a) the method presented in Sec. 3.3, using the wetness texture (b).

3.2.2 Curliness Control

We define a function $\boldsymbol{\phi}(h)$ (Fig. 2 (e)) that controls the fur curling. This function computes a displacement in the tangent space of the input mesh as:

$$\boldsymbol{\phi}(h) = \alpha(h)\cos(\omega h)\mathbf{u} + \alpha(h)\sin(\omega h)\mathbf{v}, \tag{4}$$

where ω is a curl frequency, $\alpha(h)$ is a curl radius factor, h is the normalized fur length, and $\mathbf{u}$ and $\mathbf{v}$ span the surface tangent space. Here we used the expression $\alpha(h) = (1 - \rho_{\text{mask}})K_\alpha h$, where $\rho_{\text{mask}} \in [0, 1]$ is the wetness at the strand location, and K_α is a constant representing the radius of the curl. Note that fur loses curliness as the wetness ρ_{mask} increases. Although other expressions can also be used in place of the proposed $\boldsymbol{\phi}(h)$, we found it to produce interesting results. Fig. 7 shows some examples of how this factor contributes to the fur shape.

3.3 Fur Clumping

The algorithm presented in this section aims at controlling the texture layers content directly by texture coordinates manipulation. Our target is to obtain an effect similar to wet fur clumping proposed by [Bruderlin, 2000, 2003] within the GPU fragment shader framework. To deform the fur in order to create clumps, we compute a displacement $\mathbf{d}$ in the fragment shader for each visible position on the input mesh. Then we displace the texture coordinates of that position if the region is wet. To check whether the region is wet or not, we inspect the wetness mask (Fig. 5) using the model parametrization. The displacement $\mathbf{d}$ is illustrated schematically in the cross-sectional view (clump

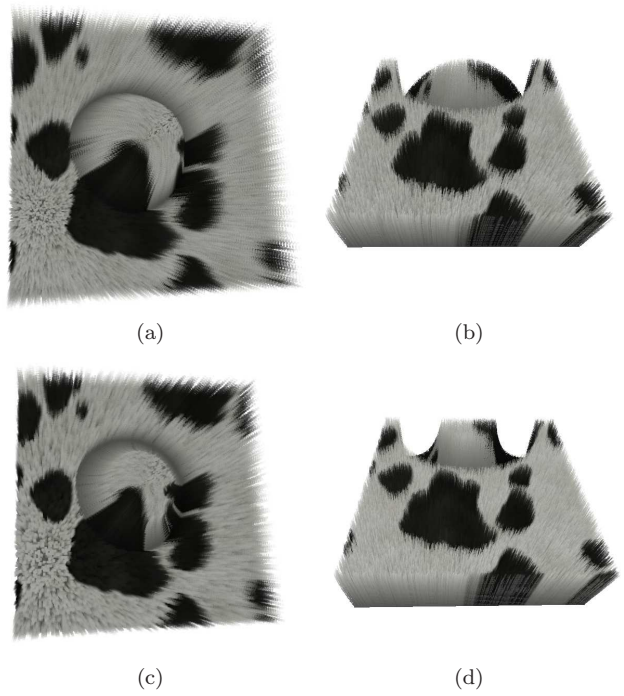


Fig. 9 Fur with $w = 0$ (a,b) and $w = 1$ (c,d).

section) in Fig. 5. We compute the displacement $\mathbf{d}$ as:

$$\mathbf{d} = ((1-w)h + wh^w)\rho_{\text{mask}} r_{\text{mask}}(\mathbf{c}_{\text{pixel}} - \mathbf{c}_{\text{mask}}), \quad (5)$$

where $h \in [0,1]$ is the normalized fur length, ρ_{mask} is the wetness, r_{mask} is the clump radius, $\mathbf{c}_{\text{pixel}}$ is the current pixel position on the wetness mask, and $\mathbf{c}_{\text{mask}}$ is the center of the clump.

The factor w is a constant that controls the blending between the functions in Eq. 5. Simply put, the factor w controls how the fur shape bends towards the clump center. Fig. 9 (a) and (b) show the illustration where the fur maintains a straight shape while leaning towards the clump center. In Fig. 9 (c) and (d), the fur shape bends in towards the center of the clump along its length. This factor is somewhat similar to [Bruderlin, 2000, 2003]'s clump-rate factor. Then, we add $\mathbf{d}$ to the texture coordinates of the wetness texture. If the ρ_{mask} read from the wetness texture (Fig. 5) at this new position is not 0 (which means the location is wet), we draw the fur using the color and fur texture coordinates displaced by $\mathbf{d}$. Otherwise we discard the fragment because it no longer comes from a clump region. If an area is not in a clump region, we do not displace any texture coordinates, and directly render the fur using the original texture coordinates.

This idea is stated in Alg. 2, where $\mathbf{uv}_{\text{color}}$ and $\mathbf{uv}_{\text{geo}}$ are the corresponding color and fur texture coordinates. Note that although the fur texture has three coordinates, only two are displaced.

Algorithm 2 Fur Clumping

```
if ρ_mask ≠ 0 then
  compute d
  if ||c_pixel − c_mask|| > r_mask then
    ignore pixel from a different clump
  end if
  if ρ_mask = 0 at c_pixel + d then
    ignore pixel from a dry area
  end if
  uv_color ← uv_color + d
  uv_geo ← uv_geo + d
  if no geometry at uv_geo then
    ignore transparent pixel
  end if
  compute color using uv_color
end if
```

4 Rendering

For rendering, the input surface parameterization is calculated using the methods from [Saboret et al., 2009]. For the fur texture creation, the methods proposed by either [Lengyel, 2000] or [Papaioannou, 2002] can be used, for which we chose the latter. The rendering process resembles previous methods like [Lengyel et al., 2001, Tariq and Bavoil, 2008]. For completeness, we do a brief description here. First, we draw the input mesh once with z-buffer writing enabled. Next, z-buffer writing is disabled, blending is enabled, and the fur texture is selected. Then we render the base surface N times, once per layer (where N is the number of layers). For each of these passes, the displacement v_{offset} is applied to each layer individually. This factor displaces the base surface to its final position.

In the vertex shader, we apply Eq. 1 to each vertex normal. In the fragment shader, we select the corresponding texture fragment from the fur texture layers (3D texture) using the interpolated surface texture coordinates (u, v) plus third texture coordinate h, which is the current normalized fur length $h = (1+n)/N$, where $n \in [0, N-1]$ is the current layer. For non-transparent fragments, we use [Kajiya and Kay, 1989] illumination model with a given specular color K_s and with the color K_{color} extracted from the input color texture, and compute the final fragment color C_{frag} as:

$$\begin{aligned} C_d &= K_{\text{color}}(\mathbf{T} \times \mathbf{L}), \\ C_s &= K_s[(\mathbf{T}\cdot\mathbf{L})(\mathbf{T}\cdot\mathbf{E}) + (\mathbf{T}\times\mathbf{L})(\mathbf{T}\times\mathbf{E})]^{p_{\text{phong}}}, \\ C_{\text{frag}} &= \gamma(C_d + C_s), \end{aligned} \quad (6)$$

where $\mathbf{T}$ is the tangent to the embedded virtual strand at fur length h, $\mathbf{L}$ is the light direction, $\mathbf{E}$ is the camera direction, and p_{phong} is the Phong exponent. In our current implementation, the tangent $\mathbf{T}$ is computed N

Table 1 Performance evaluation. In a large part of the examples we obtained real-time performance.

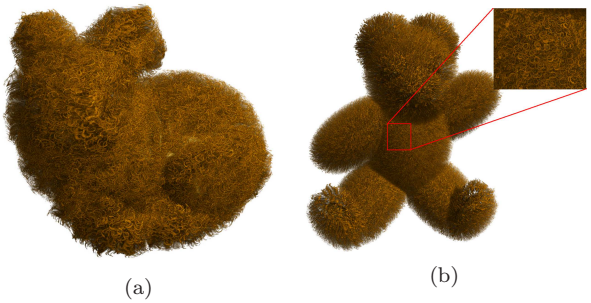
Model	#Triangles	16 Layers	
		800 × 600	1680 × 1050
Torus	9.360	267	117
Pear	20.845	256	117
Hippo	44.452	154	80
Cow	92.689	78	63
Horse	95.533	66	52
Bunny	142.307	24	22
Elephant	154.789	25	21
Lion	365.118	3	3
Model	#Triangles	64 Layers	
		800 × 600	1680 × 1050
Torus	9.360	90	35
Pear	20.845	84	34
Hippo	44.452	51	24
Cow	92.689	28	21
Horse	95.533	23	17
Bunny	142.307	8	8
Elephant	154.789	7	7
Lion	365.118	< 1	< 1

times per triangle, once for each layer. Highlights due to the fur curving are visible for example in Fig. 6 (a). Another possible choice for the color computation is using [Gupta and Magnenat-Thalmann, 2007], since it accounts for the water contained in the fur. We add a simple shadow effect by introducing a factor $\gamma = (K_{\text{shadow}} - 1 + h)/K_{\text{shadow}}$, for $K_{\text{shadow}} \geq 1$ chosen by the user, which multiplied by the fur color darkens the lower part of the fur. Although methods such as [Lokovic and Veach, 2000, Yang et al., 2008] can possibly provide better results, for short fur this simple method is effective and inexpensive. Fig. 8 (a) shows a rendering of normal (or dry) fur, which is similar to previous methods. Applying the wetness mask in Fig. 8 (b) to Fig. 8 (a), the desired clumping effect is achieved as shown in Fig. 8 (c).

5 Results

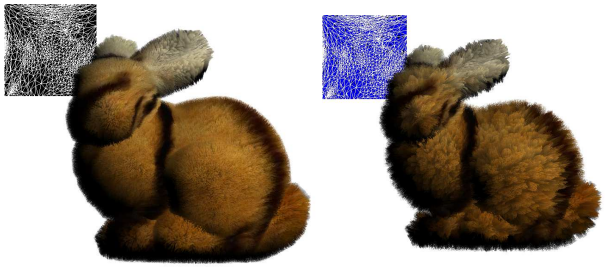
Fig. 10, Fig. 11, Fig. 13 and Fig. 12 show some more examples, which were rendered using 16 layers. Our performance evaluations were conducted on a desktop PC with an *Intel Quad Core*^{TM} $3GHz$ CPU, $2GB$ main memory, and a *NVIDIA GeForce 8800 GTS* GPU with $320MB$ dedicated video memory. The resolution of the wetness mask and fur texture layers is 256×256. The performance of our method decreases about linearly with the number of layers and sub-linearly with the screen resolution.

The frame-rates obtained are summarized in Tbl. 1. Our results indicate that as the model size (number of triangles) increases, the resolution of the screen influences less the frame-rate. On the other hand, the number of layers always affects the frame-rate. In our results, rendering at 64 layers performs at about 1/3 of the frame-rate as when rendered with 16 layers. However we still obtain real-time performance for the relevant models. When compared with previous methods (e.g. [Lengyel et al., 2001, Tariq and Bavoil, 2008]), our method adds a fur clumping effect at the expense of some additional computation.



(a) (b)

Fig. 10 (a) Stanford Bunny and (b) Teddy bear (b) covered with curly fur.



(a) (b)

Fig. 11 Stanford Bunny with (a) dry fur and (b) wet clumped fur. The wetness mask and the parameterization used can be seen on the top-left part of the figures. The black regions represent dry areas, while the blue regions represent wet areas.

6 Conclusion and Future Work

We have presented a method to add dynamic behavior to fur represented by texture layers. One part which controls the fur curliness and another part that controls the fur clumping. Our method is aimed for a GPU implementation, and is suitable for real-time applications. Our results show that our method can produce the desired clumping and curling effects for a small overhead in the rendering. We think this is a small price to pay for the extra dynamic characteristics that our method adds, which were not present in previous real-time rendering methods.

We think our method can be expanded by adding support to other styles of fur, apart from the curly or wavy kinds. Perhaps even more interesting would be

(a) (b) (c)

Fig. 12 (a) A cat head covered with dry fur, and (b) after adding random droplets over the surface. The word "CGI10" written on a furry square, and the corresponding wetness mask at the upper left corner of (c).

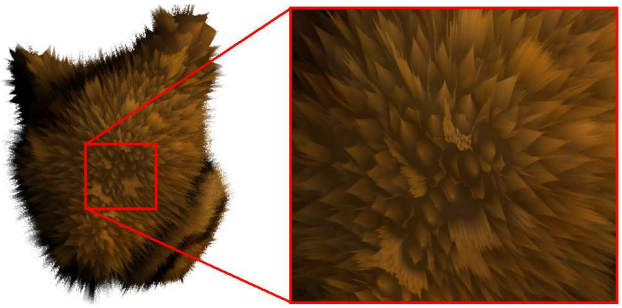


Fig. 13 Detailed view of the fur clumps.

to encode the fur shape in a color texture to facilitate the integration with the whole coloring and shape control framework. Additionally, we think that an interesting extension would be to add support to contact interactions. For example, within a physics framework, a soft-body tends to deform when collides. We think our method can be extended to deform the fur in these situations.

References

B. Bakay, P. Lalonde, and W. Heidrich. Real time animated grass. In *Eurographics 2002 Short Papers*, pages 32–41, 2002.

S. Banisch and C. Wüthrich. Making grass and fur move. *Journal of WSCG*, 14(1-3):25–32, 2006.

A. Bruderlin. A method to generate wet and broken-up animal fur. *The Journal of Visualization and Computer Animation*, 11(5):249–259, 2000.

A. Bruderlin. A basic hair/fur pipeline. In *ACM SIGGRAPH 2003 Course Notes - Photorealistic Hair Modeling, Animation, and Rendering*. 2003.

C. Csuri, R. Hackathorn, R. Parent, W. Carlson, and M. Howard. Towards an interactive high visual complexity animation system. In *ACM SIGGRAPH 1979 Conference Proceedings*, pages 289–299, 1979.

G. Elber. Geometric texture modeling. *IEEE Computer Graphics and Applications*, 25(4):66–76, 2005.

K. W. Fleischer, D. H. Laidlaw, B. L. Currin, and A. H. Barr. Cellular texture generation. In *ACM SIGGRAPH 1995 Conference Proceedings*, pages 239–248, 1995.

A. V. Gelder and J. Wilhelms. An interactive fur modeling technique. In *Graphics Interface 1997 Conference Proceedings*, pages 181–188, 1997.

D. B. Goldman. Fake fur rendering. In *ACM SIGGRAPH 1997 Conference Proceedings*, pages 127–134, 1997.

R. Gupta and N. Magnenat-Thalmann. Interactive rendering of optical effects in wet hair. In *Proceedings of the 2007 ACM Symposium on Virtual Reality Software and Technology*, pages 133–140, 2007.

R. Habel, M. Wimmer, and S. Jeschke. Instant animated grass. *Journal of WSCG*, 15(1-3):123–128, 2007.

J. Isidoro and J. L. Mitchell. User customizable real-time fur. In *ACM SIGGRAPH Conference Abstracts and Applications*, page 273, 2002.

S. Jeschke, S. Mantler, and M. Wimmer. Interactive smooth and curved shell mapping. In *Proceedings of the 2007 Eurographics Symposium on Rendering*, pages 351–360, 2007.

S. Jiao and E. Wu. Simulation of weathering fur. In *Proceedings of the 2009 International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 35–40, 2009.

J. T. Kajiya and T. L. Kay. Rendering fur with three dimensional textures. In *ACM SIGGRAPH 1989 Con-*

ference Proceedings, pages 271–280, 1989.

J. Kloetzli. *A Volumetric Approach to Rendering Microgeometry Using Precomputed Radiance Transfer*. Undergradute thesis. University of Maryland, 2006.

M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *ACM SIGGRAPH 1999 Conference Proceedings*, pages 433–438, 1999.

J. Lengyel, E. Praun, A. Finkelstein, and H. Hoppe. Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 Simposium of Interactive 3D Graphics*, pages 227–232, 2001.

J. E. Lengyel. Real-time hair. In *Proceedings of the 2000 Eurographics Workshop on Rendering Techniques*, pages 243–256, 2000.

T. Lokovic and E. Veach. Deep shadow maps. In *ACM SIGGRAPH 2000 Conference Proceedings*, pages 385–392, 2000.

M. McGuire and J. F. Hughes. Hardware-determined feature edges. In *Proceedings of the 2004 International Symposium on Non-Photorealistic Animation and Rendering*, pages 35–47, 2004.

A. Meyer and F. Neyret. Interactive volume textures. In *Proceedings of the 1998 Eurographics Symposium on Rendering*, pages 157–168, 1998.

G. S. P. Miller. From wire-frames to furry animals. In *Graphics Interface 1988 Conference Proceedings*, pages 138–145, 1988.

F. Neyret. Modeling and animating, and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):55–70, 1998.

S. Owada, T. Harada, P. Holzer, and T. Igarashi. Volume painter: Geometry-guided volume modeling by sketching on the cross-section. In *Proceedings of the 2008 Eurographics Workshop on Sketch-Based Interfaces and Modeling*, pages 1–8, 2008.

G. Papaioannou. A simple and fast technique for fur rendering. Technical report, University of Athens, 2002.

K. Perlin and E. M. Hoffert. Hypertexture. In *ACM SIGGRAPH 1989 Conference Proceedings*, pages 253–262, 1989.

L. Saboret, P. Alliez, and B. Lévy. Planar parameterization of triangulated surface meshes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 3.5 edition, 2009. URL `http://www.cgal.org/Manual/3.5/doc_html/cgal_manual/packages.html#Pkg:SurfaceParameterization`.

B. Sheng, H. Sun, G. Yang, and E. Wu. Furstyling on angle-split shell textures. *Computer Animation and Virtual Worlds*, 20(2-3):205–213, 2009.

K. Takeuchi, N. Petit, G. Guidet, and M. M. Maes. Production tools for furry characters. In *ACM SIGGRAPH Asia 2009 Posters*, page Article No. 13, 2009.

S. Tariq and L. Bavoil. Real time hair simulation and rendering on the gpu. In *ACM SIGGRAPH 2008 Talks*, page Article No. 37, 2008.

G. Yang, H. Sun, W. Wang, and E. Wu. Interactive fur modeling based on hierarchical texture layers. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and its Applications*, pages 343–346, 2006.

G. Yang, H. Sun, E. Wu, and L. Wang. Interactive fur shaping and rendering using nonuniform-layered textures. *IEEE Computer Graphics and Applications*, 28(4):85–93, 2008.




Paulo Silva received his B.E. in Electrical Engineering from the I.S.C.T.E., Portugal, in 2005, and his M.S. degree in Computer Science from the University of Tokyo, Japan, in 2010. His research interests center on computer graphics particularly on real-time rendering.

Yosuke Bando received B.S., M.S., and Ph.D. degrees in Computer Science from the University of Tokyo in 2001, 2003, and 2010, respectively. After joining the TOSHIBA Corporation in 2003, he has been engaged in the development of advanced semiconductor chips for computer graphics, vision, and image processing.

Bing-Yu Chen received the B.S. and M.S. degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1995 and 1997, respectively, and received the Ph.D. degree in Information Science from The University of Tokyo, Japan, in 2003. He is currently an associate professor jointly affiliated with the Department of Information Management, Department of Computer Science and Information Engineering, and Graduate Institute of Networking and Multimedia, of the National Taiwan University, and is a visiting researcher in the Department of Computer Science of The University of Tokyo. His research interests are mainly for computer graphics, geometric modeling, image and video processing, and human-computer interaction. He is a member of ACM, ACM SIGGRAPH, Eurographics, IEEE, IEICE, and IICM.

Tomoyuki Nishita received the B.E., M.E., and Ph.D. degrees in Electrical Engineering from Hiroshima University, Japan, in 1971, 1973, and 1985, respectively. He worked for Mazda Motor Corp. from 1973 to 1979. He has been a lecturer at the Fukuyama University since 1979, then became an associate professor in 1984, and later became a professor in 1990. He moved to the Department of Information Science of the University of Tokyo as a professor in 1998 and now is a professor at the Department of Complexity Science and Engineering of the University of Tokyo (since 1999). He received a Research Award on Computer Graphics from IPSJ in 1987, Steven A. Coons Awards from ACM SIGGRAPH in 2005, and a CG Japan Award in 2006. His research interest is mainly in computer graphics. He is a member of IEICE, IPSJ, ACM, and IEEE.