A Stylized Cartoon Hair Renderer

Jung Shin
University of Canterbury
Christchurch, NZ
jung.shin@hitlabnz.org

Michael Haller
Upper Austria University of
Applied Sciences, Austria
haller@fh-hagenberg.at

M. Mukundan
University of Canterbury
Christchurch, NZ
mukund@cosc.canterbury.ac.nz

ABSTRACT

This paper describes a new hair rendering technique for Anime characters. The overall goal is to improve current cel shaders by introducing a new hair model and hair shader. The hair renderer is based on a painterly rendering algorithm which uses a large amount of particles. The hair model is rendered twice: first for generating the silhouettes and second for shading the hair strands. In addition we also describe a modified technique for specular highlighting. Most of the rendering steps (except the specular highlighting) are performed on the GPU and take advantage of recent graphics hardware. However, since the number of particles determines the quality of the hair shader, a large number of particles is used which reduces the performance accordingly.

Categories and Subject Descriptors

I.3.3 [**Computer Graphics**]: Picture/Image Generation; I.3.3 [**Computer Graphics**]: Three-Dimensional Graphics and Realism.Color, shading, shadowing, and texture

Keywords

Cartoon Shading, Hair Rendering, Stylized Rendering

1. INTRODUCTION

Hair plays an important role in Anime, the Japanese version of Animation. The hair is not only one of the most important visual features for human beings in real life, but also for human beings in comics, cartoons, and Anime. After eyes, the hair is the feature that best shows the characters' individuality [12]. Hairstyle says a lot about personality and is characterized by the simplification of the hair strands and shading, the black outlines, and by special specular highlighting. The hair is a crucial feature used to distinguish between different cartoon characters, especially in Anime (cf. figure 1). There has been significant research on improving the photo-realistic rendering quality of hair [2, 6], however, there has been little research on cartoon hair.

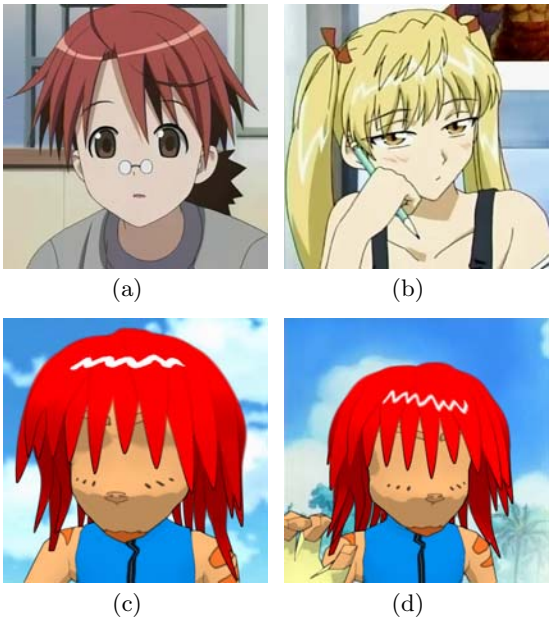


Figure 1: Real Anime examples (a-b) and results of our algorithm (c-d).

The aim of this paper is to produce high quality Anime hair images by using a sophisticated hair model. We mainly focus on:

- an easy hair model for efficient Anime rendering,
- the generation of outlines for the hair strands,
- hair shading with a modified toon diffuse shading, and
- an efficient specular zigzag highlighting, characteristic of Anime hair.

After an overview of related work, in section 2, we demonstrate our approach, including the presentation of the hair model, the creation of the silhouettes, the diffuse shading, and the charactestic zigzag specular highlighting. Note that our approach is based heavily on OpenGL extensions provided by the latest computer graphics hardware. Performance tests and results are discussed in section 4. Finally, we conclude the paper with directions for future work.

2. RELATED WORK

It is still difficult to render over 100,000 hair strands in real-time. Therefore, the hair model is an essential part of the hair shader. A good overview of different hair models are presented in [9].

Noble and Tang use NURBS surfaces for their hair geometry [14]. In [8], the hair is modeled in 2D NURBS strips, where the visible strips are tessellated and warped into U-shape strips. In contrast, Kim and Neumann use an optimized hierarchy of hair clusters [5]. Mass-Spring based hair models [16] are commonly used for hair simulation, because they are simple and efficient to use.

However, previous publications in hair modeling have mainly been focused on impressive computer-generated hair simulation and rendering with realistic and photo-realistic behavior (cf. [18]). Several papers have been published to improve the appearance, dynamic and self-shadowing of hairs [6, 2], and the model of human hairs [5]. However, fewer researchers have focused on cell-based (cartoon) hair models [10, 14]. Mao et al. present an interactive hairstyle modeling environment for creating non-photorealistic hairstyles [10]. Their system includes a user-friendly sketch interface which generates hairstyles that can be sketched by modelers. Hairstyles are simply generated by sketching free-form strokes of the hairstyle silhouette lines. The system automatically generates the different hair strains and renders them in a cartoon, cel animation style. A similar approach with good results has been presented in [17].

Our approach is based on the painterly renderer model presented by Meier in [11] where a large number of particles had been used to simulating brush strokes paintings. Silhouettes are a very important feature in many Non-Photorealistic rendering (NPR) algorithms, especially in cartoon shading. Interesting and efficient algorithms for artistic silhouettes are proposed by [4, 15]. In [19] Wilson and Ma present an algorithm where complex 3d objects are rendered in a Pen-and-Ink style using hybrid geometry and image-based techniques.

This paper concentrates on removing unnecessary detail of the Pen-and- Ink style when the underlying 3D object is complex. Current research in NPR is directed towards improving toon shading to achieve a more toon-like rendering. Nasr and Higgett introduce a new rendering shader to make rendered objects not too glossy and shiny [13]. Anjyo and Hiramitsu introduced cartoon style specular highlights to make toon shading look more like cel animation [1]. In addition, Lake et al. present a real-time animation algorithm for toon shading [7].

The most novel elements of our work include

- an efficient and easy-to-use hair model using particles rendered on GPU,
- a silhouette renderer for the hair strands, and
- the combination of diffuse lighting with zigzag specular highlighting effects.

3. PROPOSED APPROACH

In this paper, we mainly focus on the rendering of hair patches. This section gives a brief overview of the hair model and its physics. In most photorealistic hair shader techniques, hundreds if not thousands of hairs are rendered and animated. In contrast, our method uses a relatively small amount of hair strands to generate a believable result.

An overview of the overall pipeline is depicted in figure 2. The algorithm consists of three steps:

1. Creation of the hair model and hair strands by using a billboard approach,
2. Definition of the silhouette, and finally the
3. Shading the hair strands with a diffuse and specular lighting model.

3.1 Hair Model

3.1.1 GPU based Particles

For rendering the hair strands, we implemented a special particle system, where a single strand consists of multiple particles. Alpha blended particles are placed at the generated points as screen-aligned billboards. Thus, our approach generates a sparse-hair model using different particles that are connected together by using a Catmull-Rom spline (cf. figure 3). The points $p_1..p_n$ are user-defined and match the character's head. Notice that the particles are based on a spring-mass model. Moreover, we implemented simple collision detection with a sphere that represents the head. This is used when the hair model is initialized and animated.

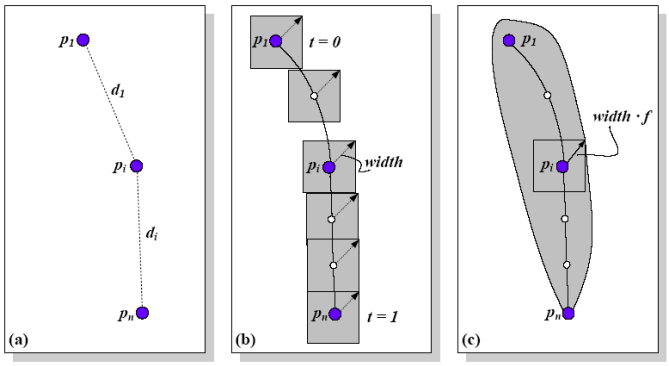


Figure 3: Creation of a hair strand: With user-defined particles (a), we define a Catmull-Rom spline on which the billboards are generated (b). Finally, the width of the billboard is re-sized to guarantee a better shape for the strand (c).

All generated particles are potential centers of the billboard particles that generate the "surface" of the strand. Thus, these particles will be used later to position the billboard texture that composes the strand. Next, the billboard vertices have to be generated. Similar to [3], this task is performed on the GPU: The coordinates of the center of the billboard - which is the coordinate of the particle - is sent to the vertex program four times, accompanied by the correct texture coordinates needed for the strand texture. These

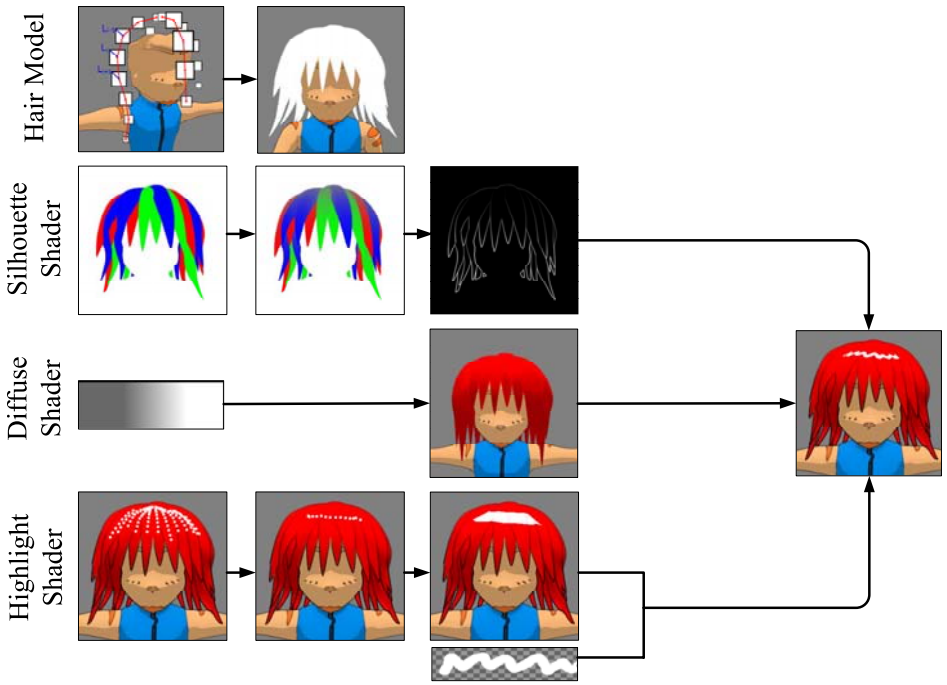


Figure 2: Overall Rendering Pipeline.

two coordinates can be used in combination to create each corner of the billboard.

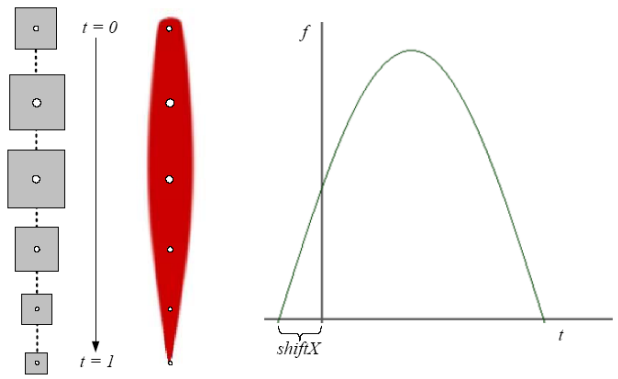


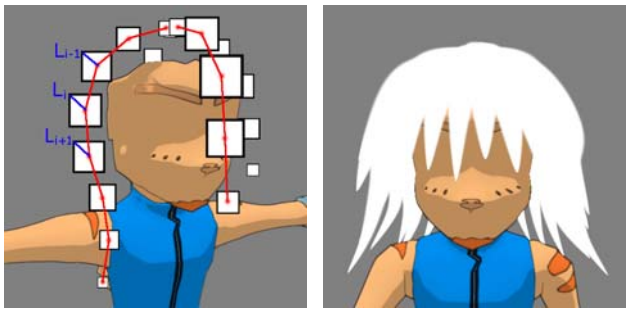
Figure 4: The billboard particles get re-sized by the factor f.

Finally, the billboard size (with its user-defined width) gets scaled by the factor

$$f = \sin\left(\frac{(t+shiftX)\cdot\pi}{1+shiftX}\right)$$

where t ranges between 0 and 1. The user-defined $shiftX$ is the shift of the sine curve along the x-axis (cf. figure 4) and has been defined by 0.2. As a result, the size of the hair-clump billboard gets scaled as depicted in figure 5.

Since the hair strands do not have a real geometry (as proposed by [10]) and are defined by billboard particles, we have

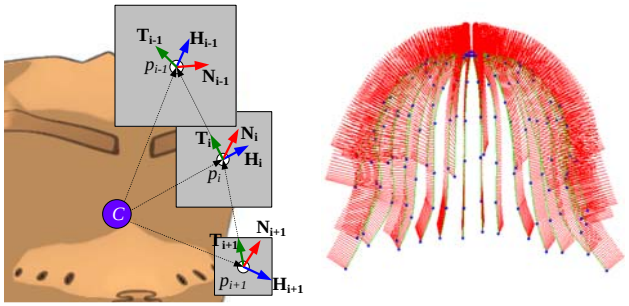


(a) (b)

Figure 5: The Hair Model: The simplified presentation of the billboard model (a). The hair model consists of 2,500 billboard particles.

to calculate the normals for further shading. Figure 6 shows how both the tangents and the normals are calculated for each billboard particle.

The user defined particle's position is defined by $p_i(i = 1, 2, .., n-1)$. Obviously, the normal of a particle has to face away from the center of the hair model. Let C be the center of the hair model and H_i is the normalized vector facing away from C the center of the head. However, H_i cannot be used as a normal vector for the billboard particle, because the particles do not necessarily have to be aligned along the head's surface. We therefore compute the tangent vector T_i which is simply calculated by $p_{i-1} - p_i$. To get the correct vector, we then calculate $R_i = T_i \times H_i$. Notice that the vector H_i has its origin in the head's center and



(a) (b)

Figure 6: Calculating the normal vector: (a) The tangent vector and the vector with its origin in the head's center are used to calculate the right vector. (b) The cross product of the right vector and the vector T_i are than used to calculate the normal vector.

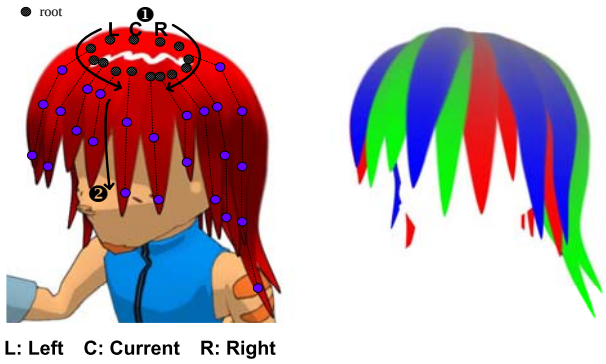


Figure 7: (a) The rendering order of the particles depending on the eye's position. (b) Relative depth order between the neighbors of the current hair strand can still influence the final rendering order of the hair strands.

is always pointing to the surface of the billboard (cf. figure 6(a). Finally, the normal vector N_i is calculated by the cross product of the normalized tangent vector T_i and the normalized right vector R_i (cf. figure 6).

3.2 Sorting the Hair Strands

After discussing the creation of the particles and the billboards, this section mainly focuses on the rendering order of the billboards. The rendering order of the strands is very important to solve the depth issues and the root of the hair strands plays an important role in determining this. We sort the hair strands according to the distance from the eye position to the root position of hair strands (cf. figure 7(a)) before rendering the individual particle billboards (b). This prevents hair strands at the back of head from covering hair strands at the front of the face.

In addition to the depth sorting, we also add a "relative" depth value, which is applied to the different hair strands. This is applied randomly at the beginning of the application and influences the final rendering order of the individual hair strands. The relative depth level (cf. figure 8(a)) is coded in RGB-color values with three depth values (R = back, G = middle, and B = front) and may change the rendering order of the hair strands. Therefore, before the current strand is rendered, the depth level of the two neighbors (the left and the right neighbor) have to be checked. The neighbors are determined by the root position of hair strands when the hair model is initialized. If the left or right neighbor is relatively behind the current hair strand (e.g. the current hair strand's color is green, but the right hair strand's color is blue), the neighbor strand has to be rendered firstly. The pseudo-code for the recursive rendering algorithm can be described as follows:

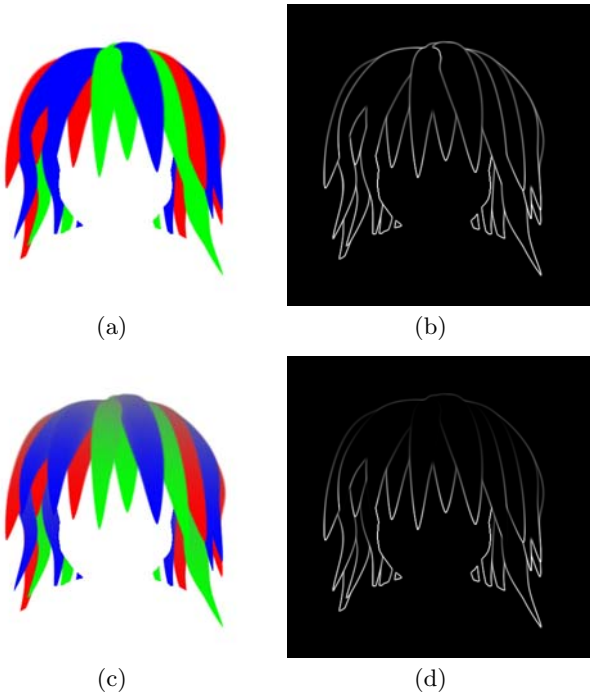
```
for each hair strand
    RenderRelativeOrder( current strand );


RenderRelativeOrder( strand ) {
    if ( strand.isRendered())
         return;
    if ( strand.left.level < strand.level)
         RenderRelativeOrder(strand.left);
    if ( strand.right.level < strand.level)
         RenderRelativeOrder(strand.right);

    render(strand);
}
```

Finally, some of the hair strands get occluded by the body's geometry (e.g. the head). The depth test between the hair billboards and the body is performed in the fragment shader. In a first step, the depth of the particles are calculated in the vertex shader.

```
vert2frag VPshader(app2vert IN) {
  vert2frag OUT;
  ...
  OUT.particleDepth = ((Particle_centerPosition.z /
                        Particle_centerPosition.w) +
                        1.0f) / 2.0f;
  ...
}
```

The reference depth map of the body is forwarded to the fragment shader, where the depth test is performed. The particle billboard has only been drawn if it is in front of the 3d geometry - thus, hair strands in the back of the head are not drawn.

```
frag2app FPshader(vert2frag IN,
                  uniform sampler2D depthRef) {
  frag2app OUT;
  ...
  refDepth = tex2D(depthRef, IN.texCoordScreen);
```



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

Figure 8: Hair Silhouettes using the relatively sorted hair strands.

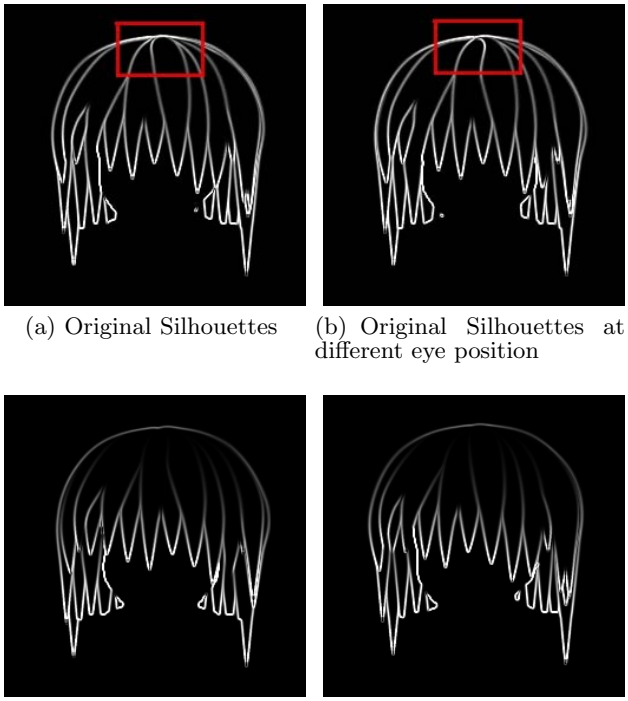
```
  // do not draw the particle if it is behind the
  // phantom geometry (e.g. head)
  if ( IN.particleDepth < refDepth ) {
    OUT.color = ParticleColor;
  }
  ...
}
```

As a result, the hair strand billboards are sorted according to their depth and relative position to their neighbor strands, and rendered into a texture for further use in the fragment shader (cf. next section). Note that the hair strands are rendered twice: first for creating the silhouettes and second for the diffuse lighting of the hair strands (cf. Section 3.4.1).

3.3 Rendering the Silhouette

Using the rendering order of the hair strands and the reference image, we can easily calculate a simplified silhouette of the hair strands. By applying a Sobel edge detection filter on the reference texture, the necessary data can be found for constructing the silhouettes (cf. figure 8(b)).

As described before, we use a two-step re-arranging approach for sorting the hair strands (in the first step the strands are sorted from the back to the front of the head; in the second step the hair strands can be slightly re-arranged according to the relative depth of their immediate neighbors). This re-arrangement of hair strands can be causing slight "jumping" effects especially near the root of the hair. This could be minimized by selecting proper width for the strands and proper relative depth.



(a) Original Silhouettes (b) Original Silhouettes at different eye position

(c) Intensity modified Silhouettes (d) Intensity modified Silhouettes at different eye position

Figure 9: Hair Silhouettes

However, a better solution is by fading out the intensity of the silhouettes which are close to the root of the hair by using a fading function. The intensity of the reference image is modified by the following function.

$$f = \frac{1.0 - cos(t \cdot \pi)}{2.0} \qquad \text{for all } t \in [0, 1]$$

Again, the variable t represents the value from the first to the last particle of a single hair strand. As the "jumping" effects are most disturbing on the top of the head, we simply fade out the hair strand on the root of the hair. The results are shown in figure 9. Figures (a) and (b) demonstrate how the silhouettes get changed, especially on the top of the hair caused by the re-arrangement of the hair strands. In contrast this effect can be hardly recognized by using the fading function (cf. figures (c) and (d)).

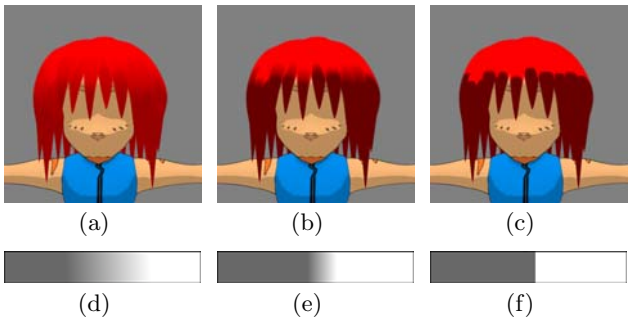
3.4 Shading

3.4.1 Diffuse Lighting

Simplification of geometry and shading is important for generating Anime-characters. Similar to the silhouettes, we use the reference image generated in the first step of the pipeline. In contrast, the order of how the particles are rendered within one hair strand is important for achieving a nice diffuse lighting effect. Therefore, the billboard particles need to be rendered from the furthest to the closest according to the eye position. To achieve a better performance,

we simply sort the hair strands according to their distance from the eye position to the root and tip of the hair strand. The pseudo code shown below uses the following notation: d_{root} is the distance between the eye position and the root of a single hair strand. d_{tip} is the distance between the eye position and the tip of a single hair strand. The function `addToList` adds the actual hair strand to the corresponding `rootList` or `tipList`.

for $i = 0$ to $n - 1$ **do**
  $d_{root} \leftarrow$ distance(EyePostion, RootCurrentHairStrand[i])
  $d_{tip} \leftarrow$ distance(EyePosition, TipCurrentHairStrand[i])
  if $d_{root} \leq d_{tip}$ **then**
    addToList(rootList, hair-strand[i])
  else
    addToList(tipList, hair-strand[i])
  end if
end for



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

Figure 10: Diffuse Lighting with Texture Lookup

After sorting the hair strands, we can render the particles. First, the color of the body's reference image is rendered, then the particles are rendered. Again the depth test is done in the fragment shader. However, diffuse shading causes a problem (cf. Figure 10(c)). The combination of the Painter's algorithm with a step-function for the texture generates unwanted shading effects. Therefore, we used a different texture to generate the diffuse lighting (cf. Figure 10(a)).

3.4.2 Specular Lighting

In [1] Anjyo and Hiramitsu present a novel highlight shader for cartoon rendering and animation. The specular highlights are an essential part of the Anime hair shading. In Asian cartoons, the specular highlight of the hair is mostly used to show the volume of the hair. We recognized that in most characters, the current specular highlighting model cannot be used. However, it does not always show the accurate shininess of hair but the approximated areas of highlights. There are many different styles of cartoon specular shapes and they are usually exaggerated and simplified. The cartoon specular does not vary much depending on the eye position.

Instead of using the traditional Blinn's specular model, we propose a new highlight shader for the 3D object. Instead, our specular term is composed by using the tangent vector T and the normal vector N of the hair strand billboard:

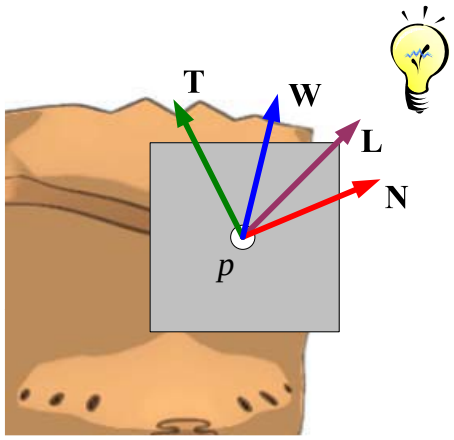


Figure 11: Calculating the specular term.

$$specular = K_S \cdot lightColor \cdot (max(L \bullet W, 0))^{shineness}$$

where L is the direction towards the light source, and W a "weighted" factor composed of the addition of the normal vector N and the tangent vector T. The vector W can be expressed by the vectors N and T, and the weight value w:

$$W = N \cdot w + T \cdot (1 - w)$$

(a) (b) (c)

Figure 12: The different weight values (a) $weight = 0.0$**, (b)** $weight = 0.5$**, and (c)** $weight = 1.0$ **can influence the position of the specular highlighting.**

Figure 11 illustrates the specular model. Notice that the weight value is user-defined and influences the final results (cf. figure 12). As a result, the highlight can be moved from the tip to the root of the hair strands by simply changing the weight value from 0 to 1.

User defined textures are used to achieve exaggerated, simplified, and cartoon-style highlights. The specular hair model has its own structure which is the same as the original hair model but containing less points. Figure 14 shows the steps to generate a stylised specular highlight.

Here we explain the merging and linking of specular points. Our algorithm iterates over all strands and removes all particle links that have a larger value than the user defined distance to each particles (cf. figure 13(a)). Consequently,

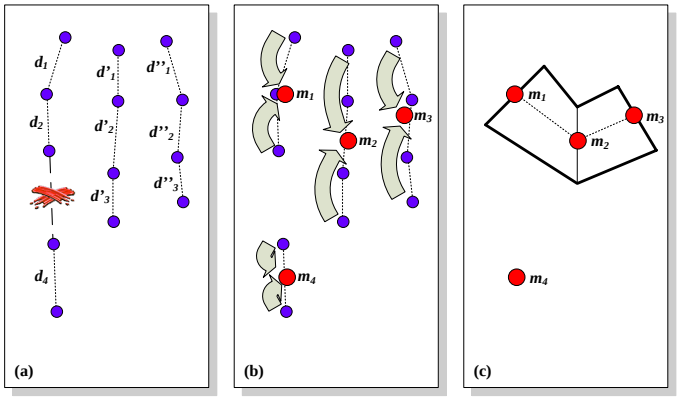


Figure 13: The pipeline of the specular highlight: after merging the particles (p_i), we achieve potential points m_i (b) which are representing potential points for creating the triangle mesh (c).

the particles of a single hair strand get merged (which is the average value of the particles of a single group) into one single particle (cf. figure 13(b)). Depending on the user-defined threshold and the distance between the single particles, one or more groups are generated. Finally, we render the highlight textures as a series of triangle strips which are composed by the merged (averaged) particles (cf. figure 13(c)).

Notice that the linked specular texture needs more than just one texture depending on the amount of particles that are connected to generate one highlight. By using a single texture, the specular highlight gets stretched and/or squished which results in unwanted artifacts. Figures 16(c) and 16(d) shows two example results of multiple specular highlight links with different lengths.

The different thresholds are defined by the user. The advantage is that the modeler can tweak the highlight to get great results. However, it may require too many user inputs (e.g. threshold value for the specular value, the minimum distance between merging points, and the linking of different textures) to generate nice renderings.

4. RESULTS AND FUTURE WORK

This system is implemented in C++ using OpenGL and nVIDIA's CG shader language. We tested our setup on a Pentinum 4, 3.00 GHz, and an nVIDIA GeForce 7800 graphics card with 256 MB. Figure 16 shows some results, where the user easily changed the properties of hair to achieve the different renderings.

The hair model was composed of 1,871 particles. However, in total we had to render 3,742 particles per frame (once for the silhouette and once for the diffuse shading). The number of sample points used for the specular highlighting contained 737 particles. The size of reference buffer was defined by the maximum resolution of 512 x 512. For the rendering of the 3D model, we used a special cartoon renderer (which improves the quality of traditional cartoon shader models) by using reference buffers and 2D filters. However, the visual improvements slow down the performance and the shader

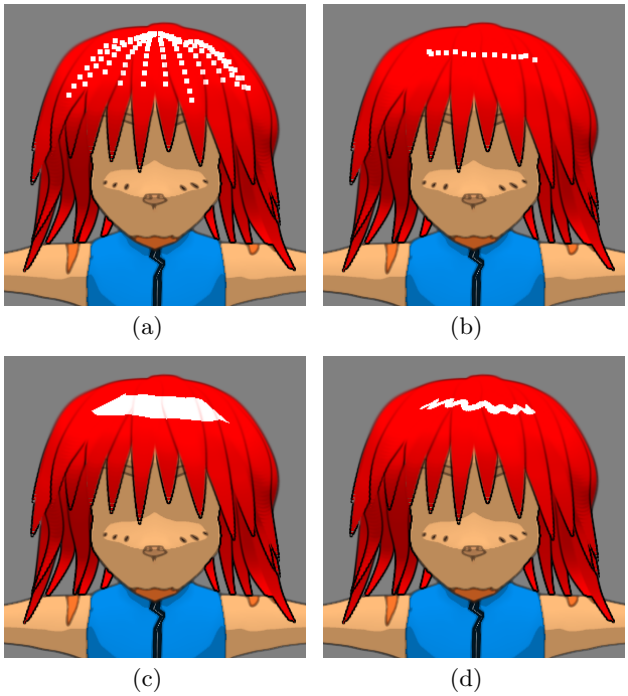


Figure 14: The different pipeline steps for generating the specular highlight: firstly, all particles are marked with a special specuar highlight threshold (a). Potential particles are merged (b) and define the triangle strip (c), which is used for rendering the highlight texture (d).

for the model (without the hair shader) runs with 7.0 $\sim$ 7.5 frames per second. The hair shader (without rendering the model) runs at 3.0 $\sim$ 3.5 frames per second. Together, both shaders are executed with 2.3 $\sim$ 2.4 frames per second. Instead of performance issues, we mainly wanted to improve the rendering quality.

This paper presents a novel rendering method for cartoon based hair models. By using a special hair model with particles, we demonstrate an efficient way to generate the outlines. Moreover, we present the lighting effects, describing how to generate both the diffuse and specular lighting. Our main goal of this study was to improve the rendering quality. However, one of the most important steps is to improve the rendering performance. Another important task is to solve the restriction of using un-twisted hair strands. Unfortunately, our approach does not allow twisted hair strands. Similar to the Painter's algorithm with teethed surfaces, we would have to split the individual hair strands. Finally, another hot topic would be the automatic animation of hair strands using a spring-mass model.

5. ACKNOWLEDGMENTS

The authors would like to thank Billy Chang for the great character model. The work is partly sponsored by the EU-New Zealand Pilot Cooperation in Higher Education under the project title "Leonardo".

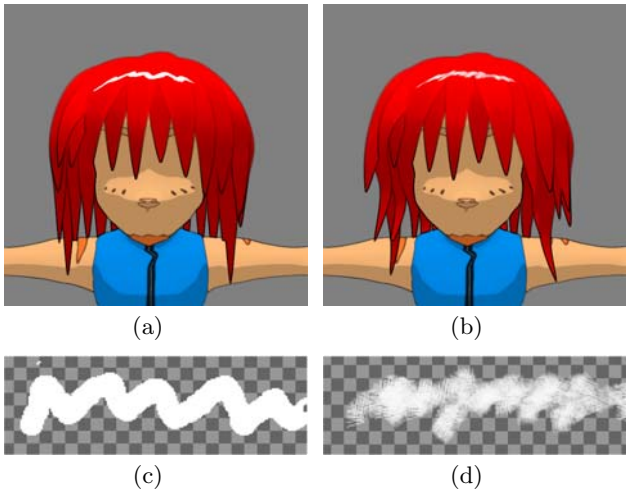


Figure 15: Stylized, specular highlighted hair texture with the corresponding texture.

6. ADDITIONAL AUTHORS

Mark Billinghurst, University of Canterbury, Christchurch, New Zealand, billinghurst@hitlabnz.org

7. REFERENCES

[1] K. Anjyo and K. Hiramitsu. Stylized highlights for cartoon rendering and animation. *IEEE Computer Graphics and Applications*, pages 54–61, 2003.

[2] F. Bertails, C. Menier, and M.-P. Cani. A practical self-shadowing algorithm for interactive hair animation. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 71–78, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[3] M. Haller and D. Sperl. Real-time painterly rendering for mr applications. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 30–38, New York, NY, USA, 2004. ACM Press.

[4] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. Coherent stylized silhouettes. *ACM Transactions on Graphics*, 22(3):856–861, July 2003.

[5] T.-Y. Kim and U. Neumann. A thin shell volume for modeling human hair. In *CA '00: Proceedings of the Computer Animation*, page 104, Washington, DC, USA, 2000. IEEE Computer Society.

[6] M. Koster, J. Haber, and H.-P. Seidel. Real-time rendering of human hair using programmable graphics hardware. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 248–256, Washington, DC, USA, 2004. IEEE Computer Society.

[7] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3d animation. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, pages 13–20, New York, NY, USA, 2000. ACM Press.

[8] W. Liang and Z. Huang. An enhanced framework for real-time hair animation. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 467, Washington, DC, USA, 2003. IEEE Computer Society.

[9] N. Magnenat-Thalmann, S. Hadap, and P. Kalra. State of the art in hair simulation. In *International Workshop on Human Modeling and Animation*, pages 3–9, 2000.

[10] X. Mao, H. Kato, A. Imamiya, and K. Anjyo. Sketch interface based expressive hairstyle modelling and rendering. In *CGI '04: Proceedings of the Computer Graphics International (CGI'04)*, pages 608–611, Washington, DC, USA, 2004. IEEE Computer Society.

[11] B. J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484, New York, NY, USA, 1996. ACM Press.

[12] H. Nagatomo. *Draw your own Manga*. Coade, 2003.

[13] N. Nasr and N. Higget. Traditional cartoon style 3d computer animation. 20th Eurographics UK Conference (EGUK '02), June 11 - 13 2002. De Montfort University, Leicester, UK, p. 122.

[14] P. Noble and W. Tang. Modelling and animating cartoon hair with nurbs surfaces. In *Computer Graphics International*, pages 60–67, 2004.

[15] J. Northrup and L. Markosian. Artistic silhouettes: A hybrid approach. In *1st International Symposium on Non-Photorealistic Animation and Rendering (NPAR'00)*, pages 31–37, Annecy, France, June 05 - 07 2000.

[16] R. E. Rosenblum, W. E. Carlson, and I. E. Tripp. Simulating the structure and dynamics of human hair: Modeling, rendering and animation. *The Journal of Visualization and Computer Animation*, pages 141–148, 1991.

[17] E. Sugisaki, Y. Yu, K. Anjyo, and S. Morishima. Simulation-based cartoon hair animation. In *13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'05)*, pages 117–122, 2005. Full Paper.

[18] P. Volino and N. Magnenat-Thalmann. Real-time animation of complex hairstyles. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):131–142, 2006.

[19] B. Wilson and K.-L. Ma. Rendering complexity in computer-generated pen-and-ink illustrations. In *3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR'04)*, pages 129–137, 2004.

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

Figure 16: Results